Office de la propriété intellectuelle du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Property Office

An Agency of Industry Canada

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings as originally filed, with Application for Patent Serial No: **2,334,375**, on February 2, 2001, by **3DM DEVICES INC.**, assignee of John Keightley, Eric Rechner and Adriano Cunha, for "Laser Micrometer ".

BEST AVAILABLE COPY

CERTIFIED COPY OF PRIORITY DOCUMENT

Agent certificateur/Certifying Officer

July 14, 2005

Date

Canada

(CIPO 68)
31-03-04

OPIC CIPO

Industry Canada Canada
File No. 1963-101-0
FEB 2 2001 5

PETITION FOR GRANT OF PATENT

1. The applicant, **3DM Devices Inc.**, whose complete address is 7-3347 262nd Street, Aldergrove, British Columbia, V4W 3V9, Canada requests the grant of a patent for an invention, entitled **LASER MICROMETER**, which is described and claimed in the accompanying specification.

2. The inventor is KEIGHTLEY, John, whose complete address is 24327 46A Avenue, Langley, British Columbia, Canada, V2Z 2M3; RECHNER, Eric, whose complete address is 453A Moodie Drive, Nepean, Ontario, Canada, K2H 8T3; and CUNHA, Adriano, whose complete address is 11776 82B Avenue, Delta, British Columbia, Canada, V4C 2E7, and the applicant owns in Canada the whole interest in the invention.

3. The applicant appoints VERMETTE & CO., whose complete address in Canada is Box 40, Granville Square, 230-200 Granville Street, Vancouver, BC, V6C 1S4, (604) 331-0381, (604) 331-0382, Canada, as the applicant's representative in Canada, pursuant to section 29 of the *Patent Act*.

4. The applicant appoints VERMETTE & CO., whose complete address is Box 40, Granville Square, 230-200 Granville Street, Vancouver, BC, V6C 1S4, (604) 331-0381, (604) 331-0382, Canada, as the applicant's patent agent.

5. The applicant believes that the applicant is entitled to claim status as a "small entity" as defined under section 2 of the *Patent Rules*.

Purpose

This document summarises the technical background behind the design of a novel large format, high speed laser micrometer.

A laser micrometer provides dimensional information about objects placed in the path of a sheet of laser light which is detected by, or scanned across a detector. The width of the 'shadow' created on the detector provides a dimension of the object.

Existing systems, 'laser micrometers', typically cover a maximum dimension of 6" at resolutions and accuracies of better than 0.001" at scan rates up to a thousand samples per second. Larger format systems, 'light curtains' are usually intended for safety monitoring to prevent human access to hazardous areas, but can provide measurements to approximately 1/8" accuracy and resolution. Typical readout rate from these larger systems is 100's of samples per second maximum.

(A sample is defined is readout of the complete detector array providing one or more measurements of object(s) in the micrometer throat)

In the smaller systems, a laser light sheet is formed using either static refractive lens elements, or a rotating mirror to scan the beam. The limitation on maximum size is the size of these light sheet forming elements.

In the larger format light curtain systems, the light sheet is actually a series of independent light beams emitted from a linear array of light emitting diodes (LEDs) spaced at the desired measurement resolution. A array of matching photodiode detectors completes the system. These designs are limited in resolution by the physical spacing between the LEDs, and limited in maximum scan rate by the need to strobe the LED's in segments to avoid crosstalk between adjacent photodiodes 'seeing' the wrong LED. The maximum scan rate is further reduced as the format increases due to the lack of a suitable interface and data encoding scheme.

The new design is intended to bring high speed and high resolution together in a large format. This is achieved by using a series of novel wide aperture laser light sheet generators as the light sources, monolithic detector arrays as matching detectors, and novel data encoding techniques to minimise the data flow back from the detectors to the central processing Hub.

The resulting design is expected to provide resolutions to better than 0.005", over widths of at least 24 feet, at scan rates of 2000 samples per second.

Key design parameters are:

Providing well collimated laser light in order to measure objects with minimal parallax error

Providing large format with single or multiple detectors

Providing a data encoding and transmission scheme capable of supporting the high data rates flowing back to the central data hub.

Development took place over the period October 1999 to the present – January 2001-01-31

The system is operational at 20 foot width, line scan rates of 2000 lines per second, resolution of 0.005" in

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

the width direction. The final form of the system is basically as described in document #1765.

The primary inventors of this system are: John Keightley, Eric Rechner, and Adriano Cunha – all current or past employees of 3DM Devices. All rights are assigned to 3DM Devices Inc.

The anticipated claims would include at least the following:

Means to provide a collimated laser light beam emitter of large format and simple construction

Means to arrange multiple emitters and matching detectors to form a large format laser micrometer

Logical means to process pixel on/off data from such detectors, assemble in compressed format and transmit to a central data processing Hub

Means to process this data and then assemble a single image or data representation of a large object passing through the laser micrometer.

Means to calibrate the position of the detectors in both x and y directions to form a seamless image

Reference Documents

The documents below are listed in historical order – oldest at the bottom of the list.

DOC refers to a document created in Microsoft Word. Many contain embedded drawings and images

DWG refers to a drawing created in AutoCAD and is always a drawing that has been used for fabrication or assembly of actual hardware.

ASSY refers to a Bill of Materials

11649 ASSY,HRPV PRODUCTION KIT LIST

11641 ASSY,EMITTER HEAD,2'

1900 DOC, PROC, HRPV EMITTER ALIGNMENT

1896 DWG,HRPV, EMITTER/DETECTOR MTG CONFIGURATION

1895 ASSY,EMITTER BASE WITH WINDOW,HRPV

1894 ASSY,BASE,DETECTOR,PARALLEL,HRPV

1893 ASSY,DETECTOR,PARALLEL,HRPV

1890 DWG,FAB,DET MTG PL,PARALLEL,HRPV

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1888 DWG,FAB,EMITTER BASE, PARALLEL, HRPV

1870 ASSY,DETECTOR,SLAVE

1867 ASSY,DETECTOR,MASTER

1866 DOC,HRPV LASER ASSY PROC

1862 DWG,ASSY, EMITTER CHANNEL,HRPV

1848 DWG,FAB,HRPV,PARABOLIC MIRROR BACK PLATE

1827 DWG,FAB,HRPV MALE CYL MIRROR BLANK

1820 DOC,LASERLINE GENERATOR EVALUATIONS

1804 DOC,HYDRA HUB HOST, DESIGN DESCRIPTION

1782 DOC,HRPV3,OPTICAL FILTER REQUIREMENTS

1779 DOC,TECHNOTE,COLLIMATOR TESTS, HRPV3

1774 DOC,TECHNOTE,HRPV3 CMOS SENSOR EVAL

1765 DOC,PLAN VIEW, LASER MICROMETER CONCEPT

1757 DOC,HYDRA HUB DESIGN, CONCEPT

Representative Existing Products from other vendors

Banner BEAM-ARRAY
Hama Sensors - Laser Micrometer

Relevant Patents from brief search - with comments -

Parabolic Mirror

4,124,277	Nov 1978	Stang (3DM integrates mount with mirror - simpler)
3,912,380	Oct 1975	Klein (3DM does not have to grind after assembly)
4,266,331	May 1981	Grassin

Light Curtains / Laser micrometers

4,432,648	Feb 1984	Musto (complex, paraboloid mirror)
4,555,633	Nov 1985	Bjorkelund (basic photoelectric measuring system)
4,402,609	Sep 1983	Fetzer (scanning type laser beam generator)
6,046,834	Apr 2000	Asada (scanning type laser beam)

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

Summary

This document describes a design concept for a large format 'Laser Caliper'. It has been developed to address the general requirements of a High Resolution Plan View (HRPV) sensor for CAE Newnes as described in their requirements specification – HRPV3_vendor spec – and previously investigated by 3DM as a potential Hydra application in technical note #1745 *Hydra – Plan View System – Feasibility Study*. There are many other potential applications of this concept in a wide range of industries.

The design concept is based on a parallel laser light sheet impinging directly onto a linear photodiode sensor. This arrangement provides the desired telecentric (parallel ray) illumination and high resolution, and is embodied in commercially available Laser Micrometers. These typically gauge to millionths of an inch over no more than a few inches of range – hence the term caliper is more applicable to the present concept which is intended to provide lower resolution over a much wider range.

Initial prototype testing has shown that the concept is capable of providing better than 0.010" resolution in length and width directions and can be configured to provide the required effective scan rate.

The proposed architecture consists of multiple linear photodiode arrays arranged in sections, in multiples of 2 feet in length, which are coupled to a central control and processing Hub. Data from this Hub is provided to the Host Decision Processor via an industry standard Fast Ethernet connection.

The immediate challenge is to design, develop, and successfully test a complete working system in the time period imposed by near term customer delivery dates – March 2000.

This document is provided as a design description to CAE Newnes and is based on the premise that 3DM will manufacture the emitter and detector modules and Hubs for installation and system integration by CAE in their scanner frames. It is not the basis for a proposal to develop such a system on a contract basis for CAE Newnes or any other customer.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

TABLE OF CONTENTS

Revision History: 1
Summary **2**

1.0 Overview **4**
1.1 Target Specifications 6
2.0 System Architecture **7**
2.1 Physical 7
2.2 Logical 7
2.3 Signal Cabling 8
2.4 Power Distribution 8
2.4.1 Emitters 8
2.4.2 Detectors 8
2.4.3 Hub 8
3.0 Emitter Head **9**
4.0 Detector head **10**
4.1 Detector Module – Slave 11
4.2 Pixel Processor 13
4.2.1 Operation Modes 14
4.2.2 Data Processing - Scanning 14
4.2.3 Resolution – Transitions per scan 15
4.3 Data Format 17
4.3.1 24-bit Transition Data 17
4.3.2 Video Data 17
4.3.3 Data Transmission 18
5.0 Hub Functions **19**
5.1 General 19
5.2 Host - Hub Command/Data Interface 19
5.3 Hub – Heads Command/Data Interface 19

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1.0 Overview

The Laser Caliper is designed to provide high resolution light curtain gauging over a large distance and a wide field of view at relative high sample rates. The principle of using opposing emitters and detectors to gauge an object as it passes through and interrupts the light beam is widely used and detectors based on this are commercially available. In the larger format (> 2 foot sections) these generally provide resolutions down to 1/8" but at scan rates of only a few tens of Hz. A significant limitation of higher resolution designs using discrete emitters and detectors is that these must be individually strobed such that adjacent detectors do not 'see' the wrong emitter. This complicates the control layout, increases the system clock rates many fold thereby imposing a maximum scan frequency, and introduces complexity in reading and reassembling the data.

The HRPV application and others require scan rates of several Khz, measurement resolution to better than 0.040", and a measuring range of tens of inches over widths to 20 feet and more. A new approach is required to economically and simply address these requirements.

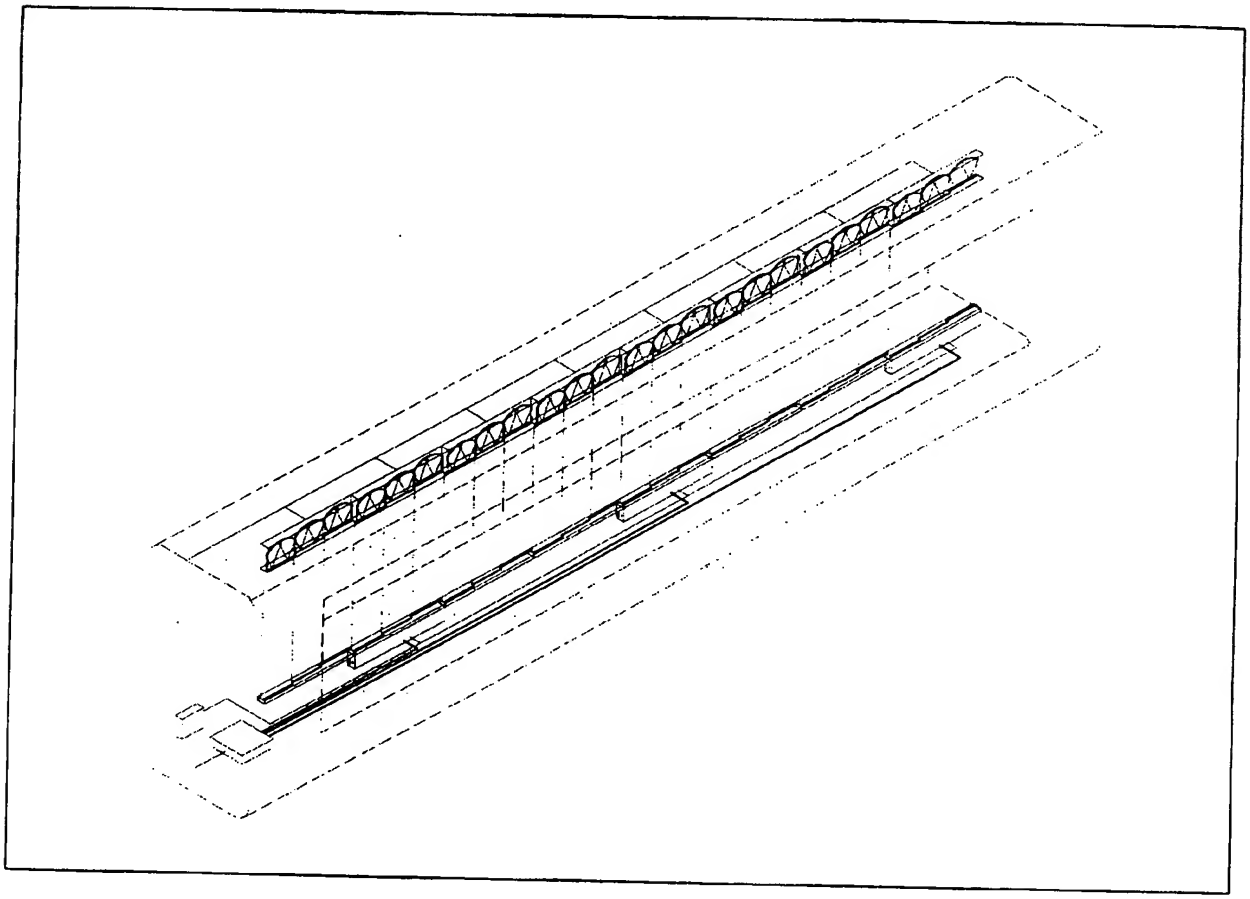
The new design concept replaces the discrete emitters with a parallel laser light sheet, and the discrete detectors with a large format linear photodiode array. The parallel feature of the laser light sheet casts a 'perfect' shadow of the target object onto the array thus removing the strobe requirement from the system design.

This concept is embodied in various commercially available 'laser micrometers'. These generally match the form factor of conventional micrometers (0-6" measuring range) but provide higher resolution and non contact measurements. Some of these devices employ a rotating scan element, the operation of others is unclear from the documentation reviewed to date. At least one commercial device is described as 'patent pending'. A brief patent search reveals the disclosure of a 'laser micrometer' as early as 1978 (US Patent #4,074,938 - Taylor) indicating that there should be no patent infringement issues with this new design.

The HRPV implementation requires a scan width of 20 feet or more, an emitter/detector separation of approximately 30", a resolution of 0.040" and a sample frequency of several KHz. The architecture to address this comprises multiple modules aligned in a single row and interfaced to a single data collection Hub. Each module is two feet in length and contains 3 linear photodiode sensors which are illuminated by three opposing laser line generators. These modules are configured as either slave or master, with each master capable of supporting two slaves - one on either side – providing a maximum of 6 feet of coverage per logical channel. A master module with none, one, or two slaves comprises a Sensor or Head. The Hub interface is capable of supporting 4 logical channels for a maximum frame width of 24 feet, although this can be extended as more than one head can be supported per channel.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

This arrangement is shown below –



The laser line generators and linear sensors are arranged parallel to the frame width. The lasers are on continuously and the detectors are operated at a fixed line scan rate of 2Khz. The system is modular in 24" sections.

Each 24" wide emitter module contains three laser line generators and beam forming optics to produce a 24" wide parallel laser sheet. Each module is powered from a central DC power supply with a third control line to turn the lasers ON/OFF for periodic background checking, and for high speed strobing to facilitate alignment during installation or replacement.

The matching detector modules comprise a master and one or two optional slave units, each of which is 24" in length, providing sensor length increments of two feet. The master and slave units are physically identical except for the absence of a pixel processor on the slave. The slaves, if present, are simply connected to the electronics on the master unit. A spare master unit could be configured as a slave for on-site backup, but given the difference in package height, it might be preferable to maintain both master and spare as site spares.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The pixel processor in each master accepts analog data from the linear photodiode arrays, digitises and extracts the edge transitions representing the board outline, and then transmits this data via fibre optic to the central Hub.

The Hub provides timing signals to all the heads and the laser units, receives the edge transition data from each, and counts encoder pulses. The Hub assembles the data, converts this to real world values, and constructs a complete 'plan view' representation of the board using the encoder data. This plan view is provided to the Host Decision Processor via a standard Fast Ethernet connection. (10baseT connection in prototype)

The Hub is described in document #1757 and is intended as a versatile control, data gathering and processing module for many applications. It provides a platform independent means to interface a variety of sensors via high speed fibre optic in a deterministic manner to a common data processing point. Data is converted, assembled and time stamped in the Hub and may then be transmitted to the Host in a non time critical fashion via Fast Ethernet.

The hub comprises four high speed duplex Hotlink fibre optic input channels memory mapped for high speed DMA transfers, an embedded PowerPC processor with VxWorks Operating System, encoder inputs and miscellaneous I/O, and an industry standard Fast Ethernet connection to the Host system of choice. All I/O is implemented as fibre optic connections except the Fast Ethernet connection which is available as twisted pair or fibre, with fibre the choice if the Hub is located exterior to the Host enclosure. The hub is AC line powered.

1.1 Target Specifications

Scanner width	up to 24 feet in two foot increments
Scanner range	16" vertical
Resolution	0.030" width and length
Sample density	0.040" length (at 80 inches per second board speed) 0.040" width
Dead band	0.050" on 24" centres (gap between sensors)
Accuracy	0.030" width and length (16' board)
Scan rate	2 KHz

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

2.0 System Architecture

2.1 Physical

The frame layout is driven by several factors –

24" increments of transverse frames

Compatibility with older frame designs to provide an upgrade option

System cost and complexity which are best addressed by providing a maximum of four channels on the Hub

Requirement to minimise inventory of spares required at site

The resulting design is as described above using 24" sensor sections with master and slave options to form 2', 4' or 6' sections to minimise the number of pixel processors and Hub channels. A future option will be the ability to daisy chain sensors to free up channels for other devices such as thickness sensors.

The emitters and detectors within each 24" section will be individually aligned during assembly to a fixed format. Alignment in the frame will be at the 24" section level.

2.2 Logical

The system will operate at a fixed sample rate determined by the time required to expose and readout a detector array. The selected sensor will be operated at a combined exposure and readout time of 0.5ms providing a scan rate of 2KHz.

All detectors will be synchronised to the same line scan reset pulse. It is expected that this reset pulse will be present on a board by board basis and provided from a separate 'board detect' photo-interrupter on the frame connected to the third fibre-optic auxiliary input on the Hub.

The emitter heads will be continuously ON during scanning. They can be turned off as required for background checking and as necessary for safety and to prolong life. The on/off switching is also designed for rates greater than 100KHz. to provide a reduced, gray scale, laser input level during system alignment.

The Cypress Hotlink interface devices support broadcast and unit addressing modes using programmable address registers and data pass through logic built in to each device. Thus it is possible to configure the Hub and detector heads network as a star, loop or combination of these.

The star configuration provides the most robust arrangement since a malfunctioning head does not fail the whole frame and it can be quickly localised. However, this arrangement is limited to four heads, unless a second bank of four Hotlink channels is added to the Hub at additional cost.

The loop configuration daisy chains as many heads as required onto a single channel. This arrangement

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

is the cheapest and simplest hardware solution but any one head failure fails the whole frame. The number of heads that can be daisy chained is limited by the available bandwidth and protocol overhead to address each head, request data, and process commands at each end of the link. Later sections in this document show a typical data rate of less than 50Kbytes per second from a complete frame – 4 sensors - which can easily be handled with all four sensors daisy chained on a single channel operating at the 16 Mbyte per second Hotlink rate.

A combination star and loop configuration could be used to lessen the impact of any one head failure. This would lead to increased software complexity to control the heads and unpack the resulting data, but is possible.

2.3 Signal Cabling

Each detector module is connected to the pixel processor on the master unit via a 20 conductor flat ribbon cable. This cable provides power to the three CIS detectors, a common clock and line start signal, a two wire I2C interface to an EEPROM for identification, control of a bi-color LED, and returns the three analog signal lines. The flat ribbon is fully enclosed by sheet metal covers.

The pixel processor is connected to the Hub via a duplex plastic fibre.

2.4 Power Distribution

2.4.1 Emitters

Power to the emitter modules will be supplied from a dedicated regulated DC supply. A three wire multidrop cable distributes this and the ON/OFF modulation signal to each module.

2.4.2 Detectors

Each master module has an internal universal AC input, DC power supply with an IEC style receptacle for input AC. This supply provides power directly to the pixel processor, and the detector modules – master and slaves – via ribbon cable.

2.4.3 Hub

The Hub is a standalone module with an internal universal AC input DC power supply. AC will be provided via a standard IEC receptacle with integral line filter.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

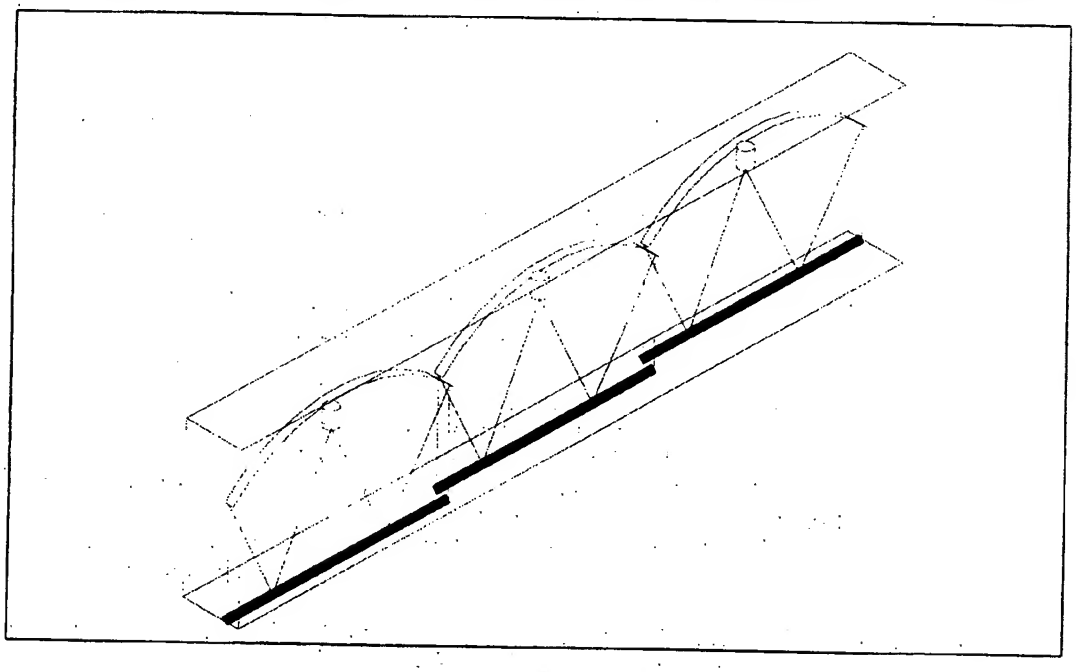
3.0 Emitter Head

Each emitter head contains three laser line generators and associated optics. The line generators are arranged in staircase configuration along the 24" length to provide overlapping coverage.

The laser is a nominal 3mw output visible (650nm) laser module with integral power supply. A collimating lens and line generating optic produce a fan shaped beam which is collimated by a parabolic mirror to form an 8.5" wide sheet of light.

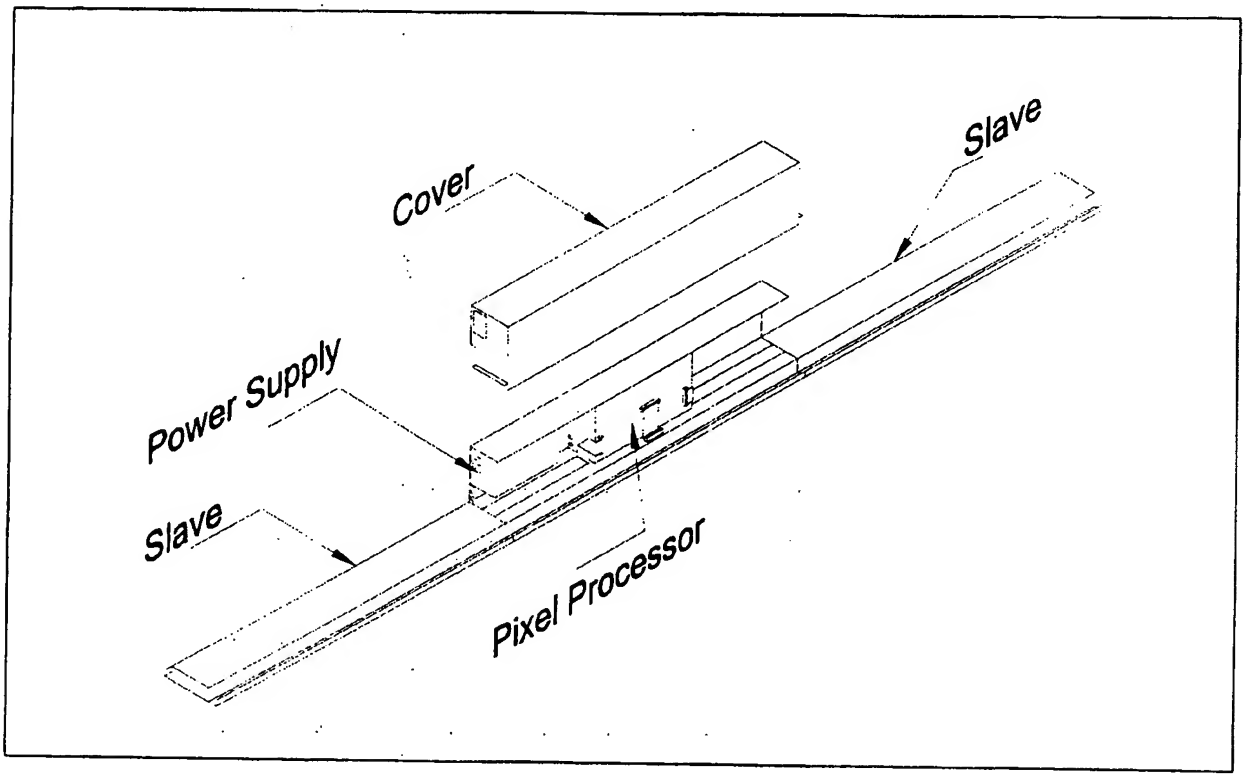
Package construction is folded steel with stiffener ribs not shown, with electrical connections located on the side of the unit. The folded laser path halves the overall height to produce a compact and stiff structure. The laser is mounted such that focus and rotation can be adjusted and the output laser sheets brought to the same factory jig standard for all heads as they are produced. There is to be no adjustment of individual lasers or lenses in the field, the only adjustment is translation and tilt of each 24" wide head in the frame to align it to the detector bank on the other side.

Each laser is nominally ON full time, but there is provision for pulsing the lasers using the control line to each unit and connected back to the hub via the power supply and fibre-optic output channel. This pulsed mode provides a much lower exposure to the detector resulting in a less than saturated signal level



This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.0 Detector head



The figure above shows the configuration of a master module and adjacent slaves – shown inverted for clarity. The slaves fit tight against the master. There is no provision for adjustment of relative positions of master and slaves as the optical system provides adequate overlap of the laser beam onto the detector for any reasonable mounting misalignment. As noted previously, one slave is flipped end-to-end, as the ribbon cable exits at one end only.

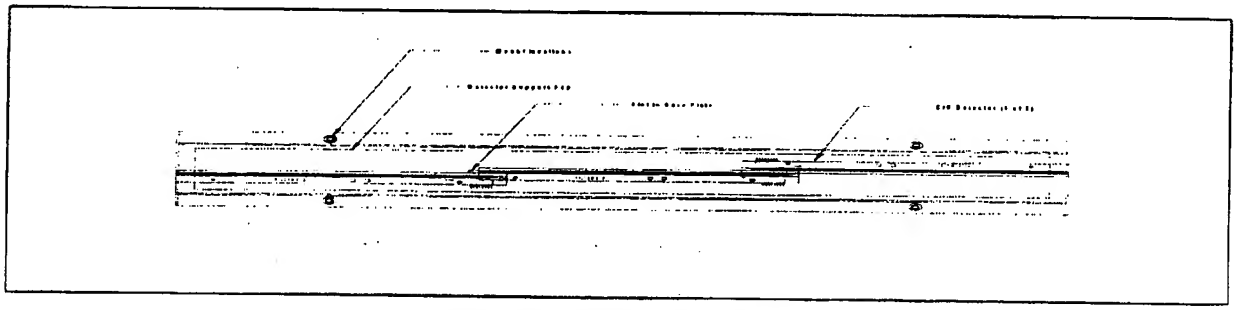
The flat ribbon cable from each slave passes straight into the master assembly and is connected to a board mounted connector on the pixel processor. The flat ribbon from the master detector module passes out through its cover and thence to the pixel processor.

The pixel processor pcb is fully enclosed in sheet metal except for openings for the ribbon cable connectors, power entry and fibre-optic lines. The power supply is a fully enclosed 'desktop' style unit with integral IEC inlet.

A sheet metal cover fits over the master unit. This cover is the only part removed for installation or change out of a master or slave unit.

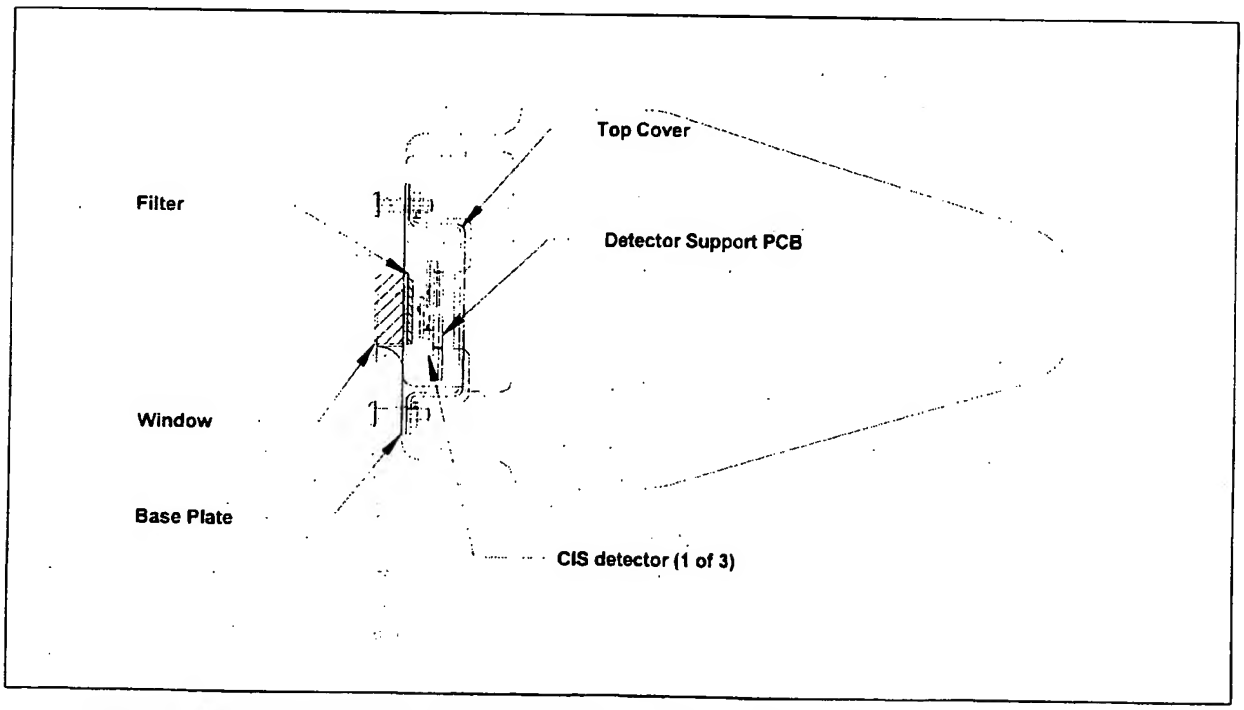
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.1 Detector Module – Slave



Detector Module – Plan View

Each 24" detector head contains three linear CIS detectors arranged with ¼" separation in staircase configuration to match the emitter beams. The detectors are mounted on a support pcb that carries a small number of components for clock buffering and a bi-color status LED that is visible to the user through the top window. The Base Plate carries a visible red filter on the underside and a thick glass cover window on the top that matches the thickness of the frame steel work. The module is sealed by the top cover with a horizontal slot at the end for the ribbon cable to exit, and is mounted to the frame with four #10-32 screws passing through its side flanges. The Slave module is shown, and this also comprises the base of the Master module.



Detector Module – Slave – Side Elevation (existing detector and frame shown for comparison)

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Each master unit has a Pixel Processor which can support the master and two slaves for a total of 9 detectors. The slaves are mounted either side of the master, with one slave flipped end-to-end to match the cable connectors – the result being that this detector is scanned left to right whereas the master and other slave are scanned right to left. The 'unscrambling' of this is performed in the Hub software.

The laser emitter 'sheets' impinge directly on the linear detector arrays. Each array is a MOS sensor with 1728 pixels at 200 dot per inch (dpi) resolution. The sensor is operated at a fixed pixel clock rate of 4 MHz and a line rate of 2Khz. Each pixel integrates continuously and is reset as it is read.

The system is operated with the laser beam saturating the detector pixels by a factor of at least 3x. This arrangement simplifies the 'image processing', does not introduce error due to blooming, and is relatively insensitive to emitter/detector misalignment.

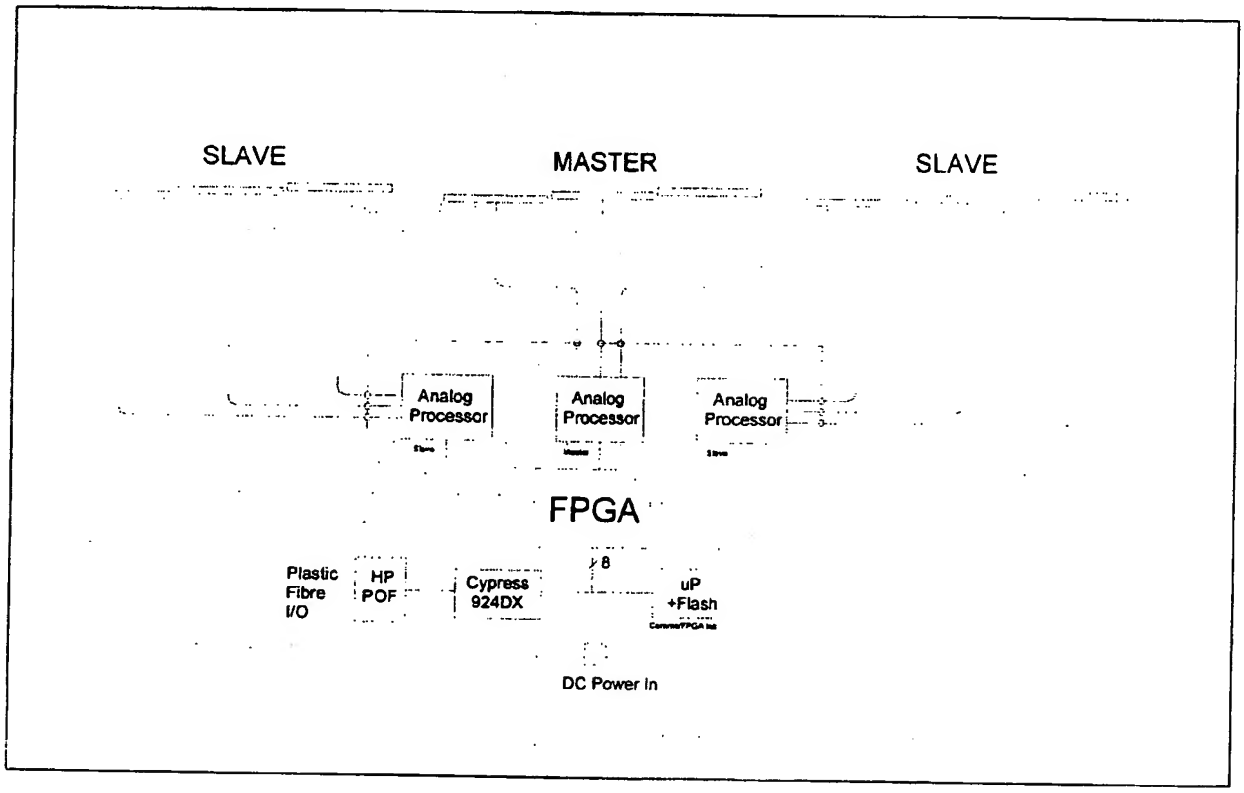
The bi-color status LED provides a visual indication of module operation as follows:

OFF	-	unit not powered and/or failed
RED	Solid	powered, not OK
RED	Flashing	powered, OK
GREEN	Solid	hub comms OK, laser full scale
GREEN	Flashing	on-line, transitions detected
ORANGE	Flashing	grey scale mode

The grey scale mode is provided for alignment of the emitter to the detector. In this mode, the laser emitters are pulsed for a fraction of a line interval and the resulting signal sent in 8bit per pixel format to the hub for display by the user application. The gray scale 'image' per detector will be nominally level with expected fall off at each end. This image is intended for continuous display during initial alignment of the emitters to the detectors to maximise the detected beam. This mode is then also available for routine diagnostic and preventive maintenance use.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

4.2 Pixel Processor



The Pixel Processor interfaces to the Hub via the Hotlink interface. It receives timing signals and commands from the Hub and returns transition data and gray scale 'video' as requested. It provides the interface for nine detectors – three on the Master unit, and three on each of the optional slaves.

The Pixel Processor is comprised of:

FPGA device (Altera Flex)
Three 3 channel CIS sensor interface IC's (Burr Brown)
Integrated Hotlink Rx/Tx with FIFOs (Cypress)
HP plastic Fibre Optic i/f
89C51 microcontroller with on-chip boot block and data flash

The FPGA implements all the digital signal processing functions. It is a Flex device which must be loaded on power up – this function is provided by the microprocessor which is also available for higher level command parsing from the Hub as necessary.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.2.1 Operation Modes

The Pixel Processor operates in the following modes –

Initialisation

Configuration

Idle

Scanning

Video

Initialisation follows power on and is a self contained operation in which the microprocessor boots from internal flash, loads the FPGA from internal flash data, queries the master and any attached slaves for presence and serial numbers, and then waits for Hotlink reframe with the Hub. Execution of the reframe sequence and establishment of communications with the Hub completes Initialisation.

The Hub performs a **Configuration** sequence in which head serial numbers are requested, and head operation confirmed.

In **Idle** mode, the head accepts configuration commands and is ready to return either scanning or video data.

In **Scanning** mode, transition table data is returned as 128 byte packets as the transmit FIFO reaches ½ Full.

The **Video** mode is used for alignment purposes. The laser duty cycle is reduced by up to 50x and the resulting gray scale video data is returned for a particular detector on request to the hub for display by the user application. This gray scale display provides an analog indication of the alignment of the laser to the detector.

4.2.2 Data Processing - Scanning

The Pixel Processor performs the following operations on the analog pixel data stream clocked simultaneously from each linear detector array.

Conversion to 8-bit digital data

Threshold comparison to a user defined value – default 128 (50% full scale) to create a serial bit stream representing ON or OFF pixels.

Removal of spurious pixels (one, two or more wide – user defined)

Assembly and writing of transition data table to Cypress transmit FIFO

The ADC task is performed simultaneously on three data streams in a single Burr-Brown device designed for processing RGB data streams.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The remainder of the tasks, all digital, are done in the FPGA.

Threshold comparison is a simple digital operation comparing the digitised pixel amplitude to a value defined in the configuration of each detector. The optical design provides a saturated detector signal and a < 20% background signal with the laser interrupted. The effect of ambient lighting is tbd but expected to be negligible with proper choice of the optical filter.

The output of the threshold comparison is either a HI or LO logic level clocked out for all nine detectors with the same pixel clock. This is encoded in the transition table that is written with a 24 bit value when there is a transition on **any** of the six channels. The most significant 12 bits contain the pixel count at which the transition occurred, and the next 9 bits the HI or LO state of each detector pixel at that instant. The least significant 3 bits are encoded as shown in the Data Formats section. -

4.2.3 Resolution – Transitions per scan

The resolution of the detectors is 200 dpi or 0.005" per pixel. The practical limit on resolution will be determined by the collimation quality of the laser sheet and spurious light affecting edge definition. Given the required resolution of 0.040", the data processing logic thus ignores every other pixel to provide a limiting electronic resolution of 0.010".

The pixel processing logic can produce large numbers of transitions in a single scan line – caused by say a badly splintered edge or 'exact' coincidence of the board edge with the line of detectors producing a gray edge which is resolved into many transitions by the comparator logic. The transition burst must not result in a buffer full state with consequent data loss for the remainder of the scan line, and/or subsequent lines until the system is cleaned up.

The Hotlink transmission logic is **not** limited to a specific number of transitions per scan line, as was the case in earlier discussions of system architecture. The system is configured such that each sensor accumulates data at a minimum rate of one transition entry per scan line, and a maximum rate many times this until the 256 byte transmit FIFO is ½ full. This results in immediate transmission of the data if there is only one sensor per channel, or transmission of data as soon as the link is available in a multi-drop network.

The real limit on system performance is the timeliness with which the Hub services the four Hotlink receiver channels, and this is determined by the loading on all four channels, not just one. The pragmatic view is that it is quite unlikely that all four channels will simultaneously produce large volumes of **valid** data, and we need really concern ourselves only with the burst capability of a single channel. The bandwidth of any one Hotlink channel is determined by –

The raw data rate of 160 Mbits/s – or 16 Mbytes/sec given 8/10 encoding

The rate at which the receive FIFOs at the Hub can be read – 32 Mbyte/sec – but this is multiplexed amongst four channels which could result in a worst case 8 Mbytes/sec per channel

System latency in processing commands

The most likely causes of excessive transitions are false triggering due to dust etc and gray edges causing triggering ON/Off – The first causes singlet transitions, the second a series of singlets followed by a steady state. A singlet is defined as a cluster of a few pixels (say 1 to 3) which should be ignored. The user can

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

define a minimum transition length, below which the transition is ignored. This limits the number of false transitions per detector, but does not necessarily avoid a buffer full condition as there are 9 detectors all OR'd together, anyone of which can require a transition table entry on any pixel clock.

The worst case situation is one transition entry for every pixel which results in the following:

½ pixel clock -> 1728/2 = 864 pixels per scan line

3 bytes per transition entry, 2KHz line rate -> 3 * 864 * 2000 bytes per second

= 5.184 Mbytes/sec

A single Hotlink channel is capable of burst data rates of 16 Mbytes/sec, and thus it would appear that the worst case scenario can be handled if there is minimal latency and competing channels in the system.

Except for large dust clouds causing false triggering, the worst case scenario is probably unreasonable given that the detectors are staggered in staircase fashion and it is unlikely that any more than three will be 'seeing' a board edge at the same time. Given this, the more likely worst case data rate is 1/3 of that above, or 1.73 Mbytes/sec.

We can conclude that any real data situation will not result in a buffer overrun condition and lost transitions. As noted in the next section, there is provision in the data format for the pixel processor to flag a buffer overrun condition to the Hub logic, and also to gracefully recover from this state using the sync count to properly tag each scan line in the transition table. Given that a buffer overrun condition was most likely caused by spurious data, the system design would appear to be reasonably adequate at this time.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.3 Data Format

4.3.1 24-bit Transition Data

Data is written to the Cypress FIFO in byte order 0,1,2

There is one 24bit entry for any Pixel Count on which one or more detectors show a transition **after** any spurious pixel filtering has been applied.

There is one 24bit entry to signify the start of a new scan line

Byte 0	Byte 1								Byte 2								
									0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Pixel count									Detector Map – wood ↓						X	0/1	0/1
															R	E	M

Pixel count is the 12bit (4096) value at which the transition takes place. This is the maximum number of pixels per detector.

Detector Map is the bit map representation of the output of each detector at Pixel Count – total of nine detectors. This representation is mapped to the same physical arrangement of detectors looking down onto the face of the detector with the direction of wood flow top to bottom of the page. In a minimal system, only the three center detectors (master head) will be active. Either or both slaves may be also active. The presence of a slave is determined during configuration. There is no change to the data format for different configurations.

The three LSB's in the 24 bit entry are bit mapped as follows –

LSB0 **M** sync pulse marker – set to I to indicate a new scan line. Pixel count is set to the sync pulse counter value incremented by Pixel Processor logic and reset to 0 on each 'new board' command.

LSB1 **E** Error indicator - Too many transitions

LSB2 **R** Reserved for future use

4.3.2 Video Data

Video data is generated at 4Mpixels/sec per detector. This is transmitted as streaming video data, one detector at a time, one line scan per burst. This is possible since the Hub requests the video data and then continually services the appropriate channel (head) until the video data has been received (or a timeout error occurs)

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The format for the video data is continuous 8bit data per pixel, sent in 14 blocks of 128 bytes for a total of 1792. There are 1768 pixels in the current sensor - the data stream is padded out to 1792+1 to properly drive the ½ full FIFO transmit/receive logic – see below.

The 1792+1 video bytes are assigned as follows:

Byte	Value	Comment
0	0x01	Start of Cell Marker. Value indicates start of video data
1	Det_Num	Detector Number 0-8
2	Ser_Addr	Head Serial Address == 0 as default
3	0x00	Reserved – packs header to 32 bit
4-1772	Video	Gray scale data – 0-255
1773-1791	0x00 (0x02)	Pad bytes*
1792	0x00 (0x02)	Start of Cell marker. Last byte

The Pad byte value is set to 0x02 if an error occurred – eg FIFO Full flag set during the line scan interval.

4.3.3 Data Transmission

The Hotlink interface is configured to transmit a fixed packet size of 128 bytes – ½ the maximum FIFO count. Data is written continuously to the Cypress transmit FIFO with at least one 24 bit value per sync pulse. As soon as the transmit FIFO is ½ full, the FPGA logic writes a Start of Cell marker with the next character to the FIFO and enables transmission which then proceeds automatically up to the Start of Cell Marker – ie 128 bytes. Note that the 128 byte packet boundary falls within a 24 bit value which is gracefully handled by the FPGA logic, at both the Head transmitter and Hub receiver state machines.

The Hub logic receives and unpacks the data from each sensor in an asynchronous fashion as each FIFO will fill at a rate based on the number of transitions.

Data gathering is synchronised by the Board Start opto-interrupter which causes a Line Start command to be sent to all the Pixel Processors. After the board has passed, the Hub logic waits an additional 43 sync intervals to ensure that each sensor has transmitted all valid data in its FIFO. (43 syncs == 129 bytes minimum == ½ full FIFO)

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

5.0 Hub Functions

5.1 General

The Hub provides a standalone platform with a built-in operating system to provide all the physical and logical interfaces to the emitter and detector heads. In on-line scanning mode, it controls the operation of the heads, receives transition data from them, assembles this into a complete plan view description and passes this to the Host on demand. In off-line mode, the Hub supports autonomous start-up of the heads, calibration, threshold determination, and various i/o functions.

Many of these functions are common to other applications and will be developed as built-in library functions. Other functions such as building the plan view description and higher level communications with the Host are application specific, the final responsibility for which lies with CAE Newnes in this instance.

The design and functionality of the Hub is covered in another document. The following sections provide a brief overview of the Hub as presently envisaged.

5.2 Host - Hub Command/Data Interface

The Hub library functions support a standard TCP/IP socket interface to a host system. A Host application based on the original LMHydra NT program provides a basic configuration, demonstration and test interface for multiple heads and will support the following operations:

Configuration of multiple heads, Hotlink network setup and diagnostics, and writing of FPGA data to heads.

Acquisition and display of grey scale video from one or more heads.

Threshold calculation and writing to heads

Calibration of single heads and 'butting' of data.

Encoder readback

The Hub VxWorks OS will be resident in flash memory and is not required to be loaded on boot.

5.3 Hub – Heads Command/Data Interface

The Hub communicates with the detector heads via the memory mapped Hotlink interfaces. Heads may be attached in star or daisy chain fashion to the Hub and an appropriate communication protocol is required in each case.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

In the star configuration, the heads are uniquely addressable by Hotlink channel and commands are routed accordingly by the software command processor and FPGA logic in the Hub. Heads may respond to commands at any time as there is no potential for collision with another head on the same link.

The daisy chain connection requires some more overhead and intervention by the Host to avoid collisions between heads when returning data to the Hub. This is done by assigning each head a unique address within its respective Hotlink channel. All heads receive the same scan start signal to acquire data, but no head returns data to the Hub until it is uniquely requested by the Hub logic. The Hub performs a round robin polling of all heads on each Hotlink channel to check for available data. Each head responds with a NACK if its transmit FIFO is less than ½ full, or proceeds to transmit 128 bytes when polled. The NACK response is a stretched TxINT received as RxINT by the Hub.

The Hotlink receive and transmit FIFO's are 256 bytes deep at each end of the link. The packet size per transmission has been fixed at 128 bytes so that the following is true:

Each head always writes 128 bytes of transition data to the transmit FIFO

The Hub DMA routine always reads 128 bytes from each receive FIFO

The scan start signal is generated by the Hub and distributed to all heads on the Hotlink using the TxINT/RxINT toggle feature in the Cypress transceivers.

The Hub may send commands to heads as either broadcast to all heads, or uniquely addressed to a single head. There is a small command set to support the following requests from the Hub:

Enumeration sequence – receive new address

Report head serial number

Report status

Scanning Mode ON/OFF

Send Gray scale video

Receive new FPGA data

Command formats are described in the Hub design document 1757.doc

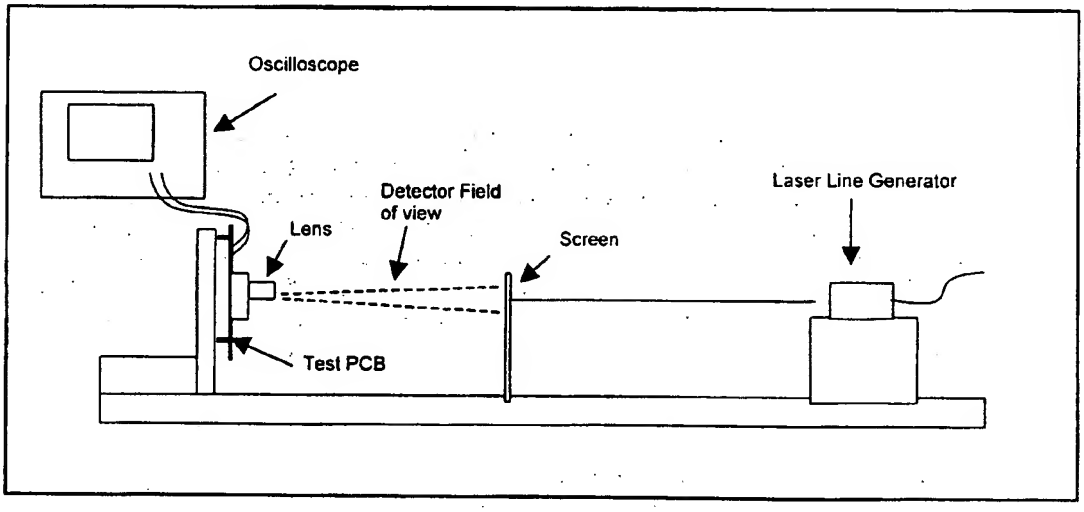
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1.0 Summary

The HRPV3 detector design employing the Photon Vision LIS1024 CMOS detector with lens and imaging screen has been evaluated using a simple test fixture. Various options for screens were investigated, and a fast f/2.0, 3.6mm lens was used for the optic. In summary, the detector does not offer adequate sensitivity for the HRPV3, and would require a laser source of at least 120mW per 12" section, resulting in high emitter costs due to the required high laser power. It should be noted that although the integration time could be increased (currently running at 3MHz), there is not enough margin to make this approach viable with moderate laser power.

2.0 Test Setup

A test fixture was constructed to simulate the HRPV3 geometry. The fixture included a test PCB for the CMOS array with a lens mount above the detector, a candidate lens (f=3.6mm, f/2.0 from Westech Optical), a 45 degree 7mW uniform intensity line generator (3DM laser), and several types of screens including opal glass, FR-4, red acrylic with diffuser, and paper. The test PCB provided access to video output signals for analysis using the digital oscilloscope. The test setup is illustrated below.



Note that in the above figure, the CMOS array is oriented such that the pixels are aligned horizontally (plane out of the page), and similarly, the laser line is oriented parallel to the array length so that the line length is imaged along the length of the array.

3.0 Discussion/Results

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The summary results are presented below. Amplitude refers to average signal level as displayed on the oscilloscope.

Screen	Amplitude (mV)		Laser Power (mW / inch)	Image quality
	B/G	Line		
Paper	150	300-500	9.3	Uniform line
FR-4	150	300-600	9.3	Uniform line
Opal Glass	150	300-500	9.3	Uniform line
Red Acrylic with diffusing tape	150	300-2500	9.3	Non-uniform, direct transmission component

Note: saturation level of the above setup is approximately 4.0V.

From the tabulated results, it appears that the laser source must be at least 112mW for a 12 inch length to obtain the modest levels indicated. It would be desireable to increase the signal levels further in order to obtain a satisfactory signal to noise ratio. This would require a laser source of a few hundred milliWatts, the cost of which would be prohibitive.

4.0 Conclusions

The HRPV3 detector design employing the Photon Vision LIS1024 CMOS detector with lens and imaging screen is not a viable approach for the HRPV3 due to the low sensitivity of the detector. In order to achieve adequate signal to noise ratio, the emitter would be required to produce a 12" laser line at a few hundred milliWatts of power. The cost of such a source would exceed the cost target for the HRPV3.

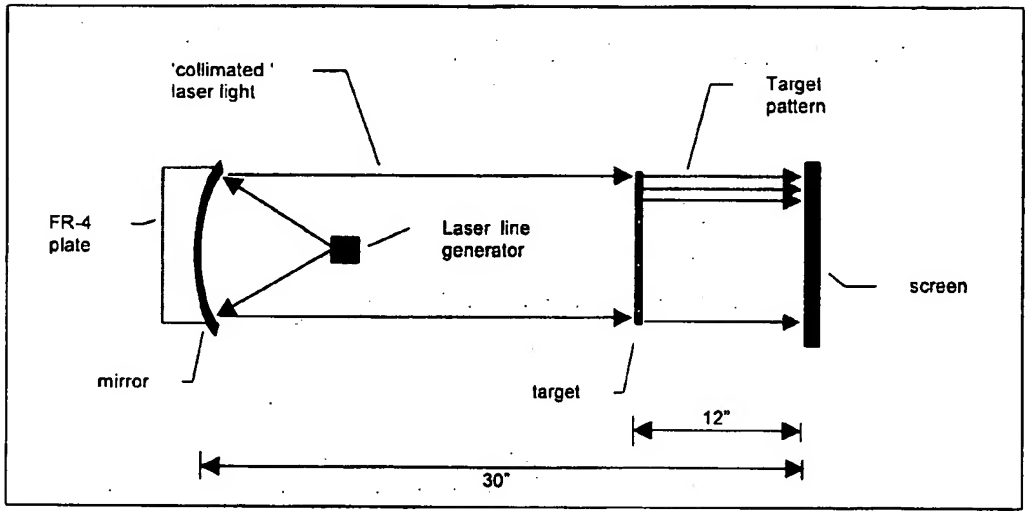
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

1.0 INTRODUCTION

The proposed HRPV collimator consists of a thin first surface mirror which is formed against a 20" radius cylinder or parabola. The mirror is secured to a 1/16" FR-4 backing plate using Durabond E-20NS epoxy. Two methods of making this mirror are presently feasible: vacuum forming against the appropriate form, and clamping to the form by securing end points of the mirror only. The resulting mirrors were evaluated for collimation performance and the corresponding error plots are given.

2.0 TEST PROCEDURE

The test setup is given in the figure below.



The laser line generator (kodak lens, mitsubishi 685nm 50mW diode) is set at a distance from the mirror for best collimation, near the focus of 10". The collimated laser light illuminates the target, and the resulting shadow of the target is viewed against the screen. The target consists of a packaging strip of surface mount resistors which gives a line of 1mm holes spaced at 4mm apart. The 'screen' consists of a vertical foamcore target with an attached plot of vertical lines spaced 4mm apart against which the shadow of the target can be directly evaluated. The deviation from true collimation is recorded, and then the target is shifted laterally by 2mm to increase the resolution of the measured collimation performance to an effective sampling interval of 2mm. The corresponding error plots for several mirrors are given on subsequent pages.

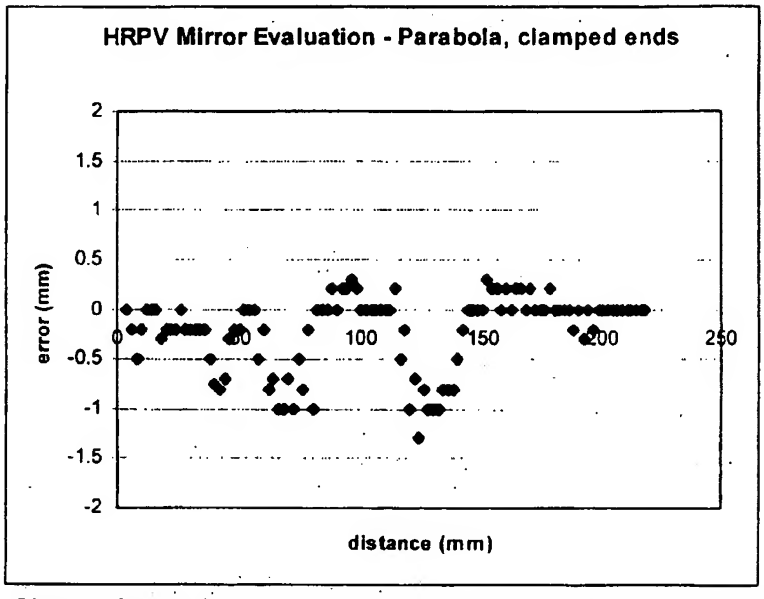
3.0 CONCLUSIONS

The collimation performance of the fabricated mirrors appears to be satisfactory for the requirements of the HRPV. Errors are within +0.8mm to –1.3mm for a 12" target-detector standoff. The best collimation performance is achieved using the parabolic form and securing only the endpoints of the mirror before fixturing to the backing plate. This method gives errors within +0.3mm to –1.3mm.
The preferred method for manufacture is to use 'vacuum hold down', however, the present form (especially the parabolic form) appears to introduce additional collimation errors due to imperfections in the form. It is suggested that the parabolic form be improved in order to achieve better collimation for a vacuum-held mirror.

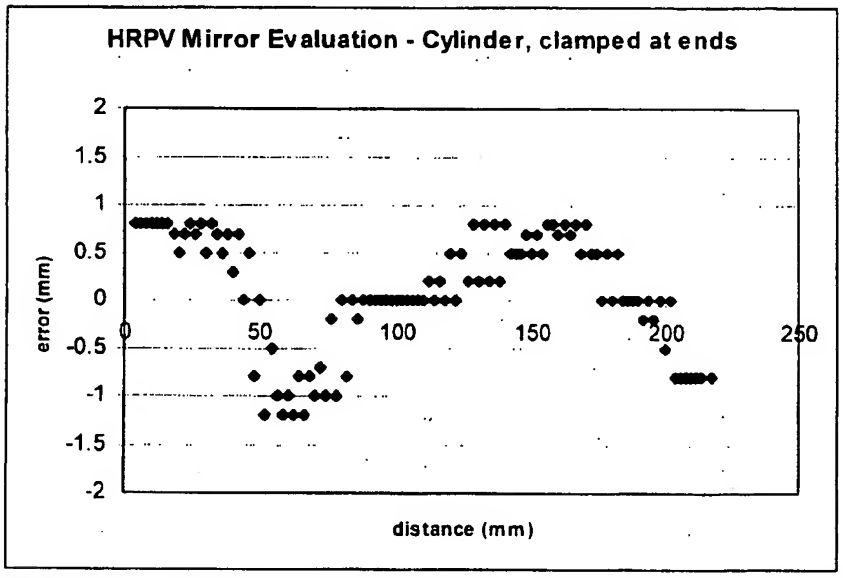
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.0 COLLIMATION ERROR PLOTS

The collimation plots are given below, with a brief observation note regarding the uniformity of the collimated laser line.

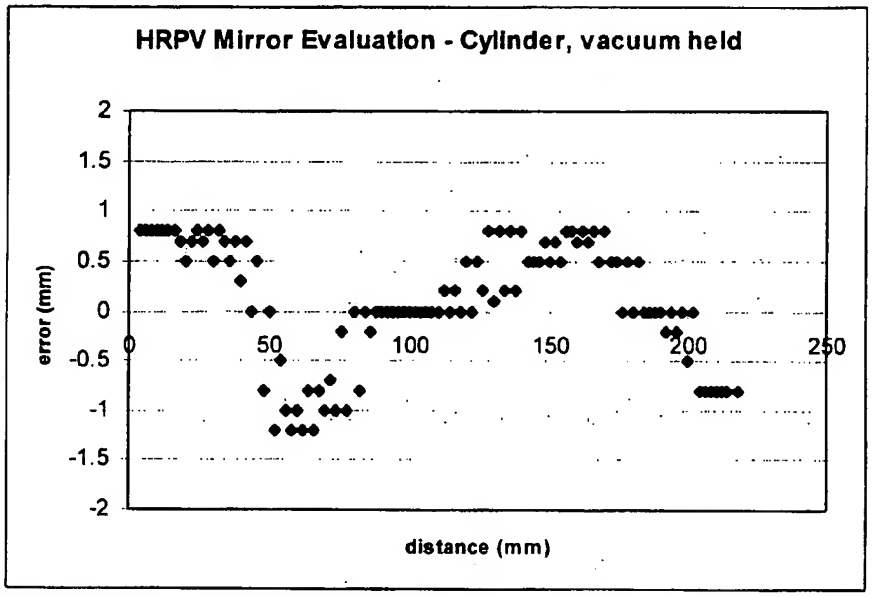


Observations: Collimated light is quite uniform, a few 'hot spots' are apparent

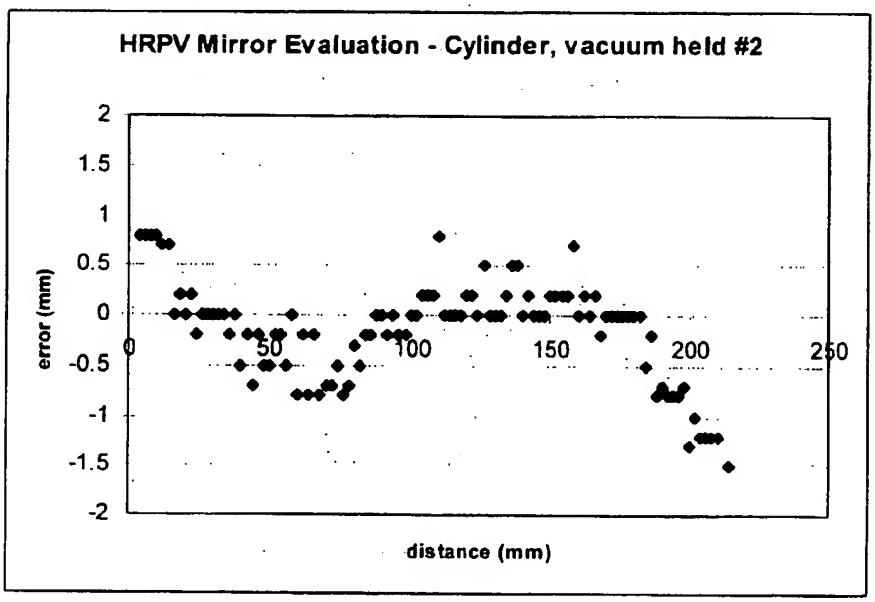


Observations: Collimated light is very uniform, no 'hot spots', a few 'warm' areas are apparent.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

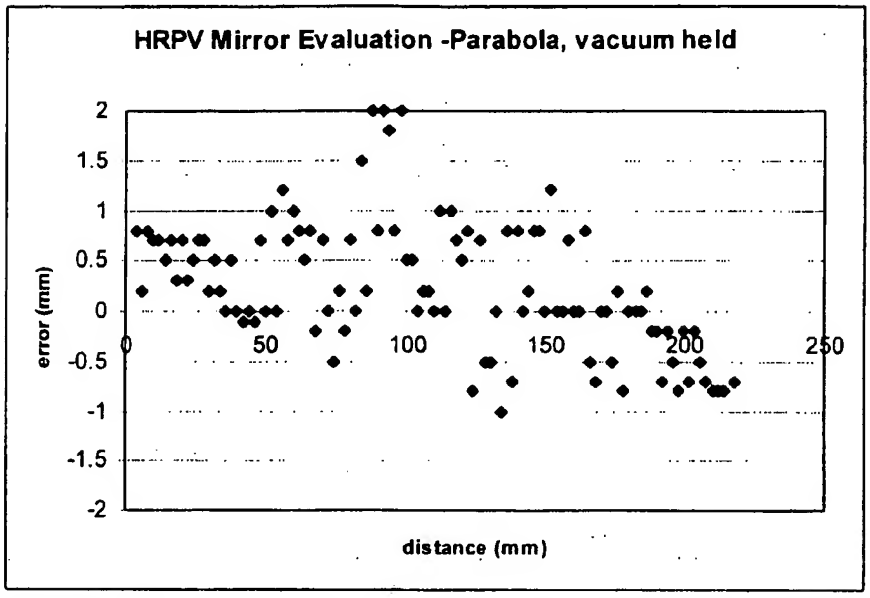


Observations: Collimated light is quite uniform, no 'hot spots', a several 'warm' areas are apparent.



Observations: Collimated light is quite uniform, with a few 'hot spots' and some 'warm' areas.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.



Observations: Collimated light is somewhat uniform, with several 'hot spots', and a few 'warm' areas.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

1782	3DM Devices - Technical Note - HRPV Scanner	18 Feb 99
	HRPV3 DETECTOR FILTER REQUIREMENTS	2

1.0 Laser Source

Candidate lasers for the HRPV are as follows:

1. Toshiba TOLD9442MDA 650nm (645nm – 655nm) 5mW
2. Samsung SLD65018260 650nm (640nm – 658nm) 5mW
3. Toshiba TOLD9221MDA 670nm (660nm – 680nm) 5mW
4. Toshiba TOLD9225MDA 670nm (660nm – 680nm) 10mW
5. Samsung SLD670182(50,60) 670nm (660nm – 680nm) 5mW
6. Mitsubishi ML1013R 685nm (670nm – 700nm) 50mW

Temperature changes will affect the laser wavelength. Typical changes are 0.25nm/C (diode case temperature), therefore the wavelength will be lowered by 8.75nm at –10C and increased by +6.25nm at 50C.

2.0 Filter Specification

Based on the laser sources listed above, the approximate filter specification can be determined as below.

Diode	Power (mW)	Wavelength Range (diode spec) nm	Thermal Component	Bandwidth Range	Approx. Filter Spec
TOLD9442MDA	5	645-655, 650ctr	-8.75, +6.25	636.25-661.25	648.75, 25 HPBW
SLD65018260	5	640-658, 649ctr	-8.75, +6.25	631.25-664.25	647.75, 33 HPBW
TOLD9221MDA	5	660-680, 670ctr	-8.75, +6.25	651.25-686.25	668.75, 35HPBW
TOLD9225MDA	10	660-680, 670ctr	-8.75, +6.25	651.25-686.25	668.75, 35HPBW
SLD67018250(60)	5	660-680, 670ctr	-8.75, +6.25	651.25-686.25	668.75, 35 HPBW
ML1013R	50	670-700, 670ctr	-8.75, +6.25	661.25-706.25	683.75, 45 HPBW

The specification is approximate because it does not include the manufacturing tolerances, where typical tolerances are +/-10-20% of the HPBW on the centre wavelength, and +/-10-20% of the HPBW on the HPBW. It is important to note that these tolerances can be combined in any manner. Thus, to cover the tolerances, it is necessary to increase the HPBW by 3 x (10-20% of approximate HPBW). The resulting specifications are given below.

TOLD9442MDA	648.75 ctr	32.5-40.0 HPBW
SLD65018260	647.75 ctr	42.9-52.8 HPBW
TOLD9221MDA	668.75 ctr	45.5-56.0 HPBW
TOLD9225MDA	668.75 ctr	45.5-56.0 HPBW
SLD67018250(60)	668.75 ctr	45.5-56.0 HPBW
ML1013R	683.75 ctr	58.5-72.0 HPBW

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

3.0 Major Consideration – Ambient Lighting

The ambient lighting is the major concern with regards to noise sensitivity and sensor performance, i.e. signal to noise and edge determination. The main types of sawmill lighting are: fluorescent, Halogen Arc, Multi-Vapor, High pressure Sodium, and incandescent. With the exception of incandescent light, all these sources exhibit a sharp cutoff near 650nm, and minimal power in longer wavelengths. Incandescent light, on the other hand, gives peak output beyond 750nm. Therefore, if the laser wavelength is limited to above 650nm (e.g. 670nm, 685nm typical), and the visible spectrum up to the range of the laser source is filtered, then incandescent illumination is the only concern. Tests have shown that a saturated detector signal caused by a 40W incandescent lamp 24" directly in front of the detector can be reduced to a low level (below 50% of saturation) simply with a near IR cutoff filter. It appears that a cost-effective filter would comprise of some kind of near IR blocking filter and a deep red wratten (e.g. wratten #70) or IR transmitting plastic filter which begins to transmit near 650nm.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1. OBJECTIVE

The HRPV emitter optics are to be aligned using this procedure to obtain a collimated 8.5" wide sheet of laser light.

2. INITIAL REQUIREMENTS AND PREPARATIONS

Required Parts/Tools

- HRPV emitter assembly kit P/N 1862
- Optical Table, emitter alignment fixture, 3 precision spacers, and alignment targets (these are to be replaced by a single fixture in future, most likely a small frame section with the typical scanner geometry, and will include a functional HRPV detector, and interchangeable targets)
- Laser Power Supply (5V, 100mA DC min.)
- Parabolic mirror assembly P/N 11640

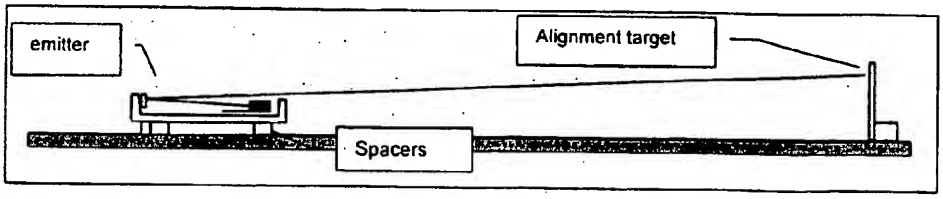
3. GENERAL PROCEDURE

The emitter assembly is to be complete, with the exception of the parabolic mirror and final alignment, i.e. all spacers are to be inserted under the laser plate, the laser is to be secured to the laser plate, etc. The emitter alignment consists of aligning the laser plate angle and position, and adjusting the parabolic mirror assembly to obtain a collimated sheet of light perpendicular to the emitter base length direction.
This procedure should be completed in no more than 10minutes for each emitter.

3.1 Aligning the Laser Plate

The laser plate must be aligned close to the correct, final angle. The final position and angle of the laser plate will vary slightly due to differences in the parabolic mirror assembly (e.g. backing plate/mirror angle) and in the case of slight angular offsets in the flat mirror on the emitter channel. For efficiency, a batch of several emitters should be aligned at a time.
Time required for this stage is approximately 2 minutes per emitter.

- Adjust the laser plate position for a nominal centred position (screws in centre of slotted holes), with no skew angle to the emitter body, and secure the two rear screws leaving the 3rd adjuster screw loose.
- Place the emitter body on the 3 precision spacers with the body oriented so that the flat mirror (folding mirror) is on the left hand side. In this orientation, the laser light is reflected off the folding mirror and proceeds to the right hand side away from the emitter toward the alignment target (as shown below).
- Place the alignment target at 29" range from the base of the emitter (see below)



- Power on the laser

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

- Tighten the adjuster screw until the laser line is at the 75mm reference line on the alignment target (75mm height above the optical table).
- If there is not enough tension in the adjuster for this position (too loose), or if there is not enough adjustment to reach this position (too tight), place the emitter aside for review. It is likely the laser plate bend angle which is causing the problem.

    - *Alternatively, try one of the following combinations of spacers for the adjuster.*
      (note: these situations are likely due to inconsistencies in the bend angle of the laser plate. If changes in the spacer are required in more than 10% of units, check the laser plates and lasers for accuracy)

Washers	Bellevilles	Thickness (mm) (belleville uncompressed)
2	1	3.10
2	2	4.08
[illegible]	[illegible]	[illegible]
3	2	5.18
4	1	5.45

3.2 Aligning the Parabolic Mirror

The parabolic mirror is to give a well collimated (but slightly diverging) laser sheet which is perpendicular to the emitter base. There should also be no skew in the plane of the laser sheet. The procedure below ensures good collimation and alignment.
Time required for this alignment is approximately 5min. per emitter.

Mounting the emitter onto the fixture and aligning the target.

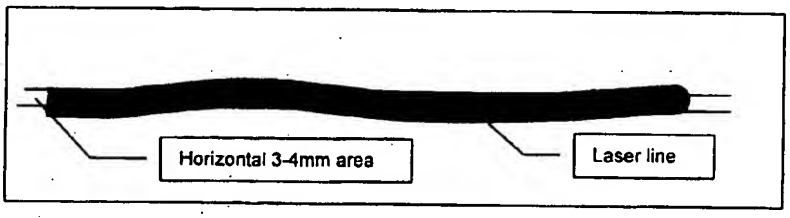
- Bolt the emitter to the alignment target, with the emitter biased to the top of the slotted holes.
- Place the alignment target onto the optical table at 30" from the emitter, and centre the target with the centre of the emitter (see centreline on optical table).

Checking the Parabolic Mirror (ensure the mirror has been destressed – black dot - and the ends trimmed)

- Power on the emitter, and hold a parabolic mirror assembly on the emitter noting the vertical location of the laser line.
- Remove the parabolic mirror assembly, and repeat the previous step with the mirror flipped 180 degrees end to end. Note the vertical location of the laser line in relation to the previous position. If there is a significant offset between the observed line locations for the two positions (e.g. >20mm), then the mirror is not perpendicular to the backing plate mounting surface. Choose the orientation which gives the straightest laser line (usually the position which gives the lower vertical position).
- Check that the laser line illuminates the central portion of the parabolic mirror. The absolute minimum requirement is that the laser line falls 1mm inside the edge of the parabolic mirror.
- If the laser line is not straight (often the case), check that the laser illuminates a horizontal region of at

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

least 3-4mm thick as shown in the figure below. If the laser line is distorted beyond this range, place the mirror aside for review or rework later.



- Mount the mirror loosely in the orientation chosen above, and proceed to the adjustment step below.

Aligning the Parabolic Mirror – Collimating and Steering the Laser Sheet

The linear (forward/backward) position of the parabolic mirror adjusts the collimation of the laser light, while the lateral (side/side) or angular position (skew) adjusts the centre of the laser light and the skew angle of the laser sheet. This step of the procedure ensures that the laser light will be collimated, centred at the target, and not skewed at an angle.

- As a starting position, pull the parabolic mirror assembly back against the mounting screws and check the shadow of the alignment fixture pins at the target.
- Move the mirror forward until the shadows of the fixture's pins overlap the marks on the target at +/- 0.5, 1.0, 2.0, 3.0, and 4.0 inches.

  NOTE: If the outer shadows are outside the marks i.e. away from the centre of the target, then the mirror must be moved back away from the laser. If there is no play in the mirror to achieve this, do the following:
  - Note the position of the laser line on the vertical target.
  - Loosen the laser plate and move the laser plate away from the mirror by approximately 2mm, ensuring there is no skew between the laser and the emitter.
  - Re-adjust the laser plate angle to obtain the same vertical position as prior to moving the laser plate.

  Alternatively, if the mirror needs more adjustment towards from the laser than is permitted by the play in the mirror, then the laser plate must be loosened and moved towards the mirror.

- Adjust the mirror laterally and/or with a skew so that the centre shadow is on the centre mark (within +/- 0.5mm), and the outermost +/- 4" shadows are approximately 1-2mm outside the marks on the target (this gives slight divergence, which is desired in the HRPV).
- Secure the mirror with the mounting screws, observing any distortion in the laser line. Once again, if when securing the mirror tightly with the mounting screws the laser line becomes distorted, check that the line is within the prescribed limits. Otherwise, place the emitter aside - shims may need to be added to compensate for the likely bend or twist in the mirror plate.

3.3 Alignment and Collimation Verification

The aligned emitter is checked for collimation and for angular deviation.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

NOTE: Currently the alignment fixture is not square to the optical table – i.e. a boresighted collimated line appears inclined so that the laser line is centred at a height of 71mm.
Time required for this check is approximately 2min. per emitter.

Angular Deviation Check

- With the emitter bolted on the emitter alignment fixture, ensure the target is placed at 30" range. The laser line should be centred at an elevation of approximately 71mm +/- 1mm on the target (highlighted pink line indicated the correct elevation).
- If necessary, adjust the collimated laser line elevation with the adjuster screw.

Collimation Verification

- Place the HRPV alignment target at approximately 30" from the emitter base.
- Place the collimation verification target (strip with ~4mm hole spacings) 12" in front of the alignment target so that the laser line projects through the holes.
- While observing the pattern of the projected holes (pattern of bright dots) at the alignment target, adjust the verification target (or the alignment target) side to side until the bright dots are superimposed on the corresponding vertical dashes of the alignment target.
- Check the pattern of the bright holes along the length of the laser line against the marks on the alignment target and ensure the following:

Position Along Collimated Laser Line	Specification
Centre	Bright dots within +/- 0.25mm of marks on target
Ends (+/- 1")	Bright dots diverge (0.5 to 1.0mm) against marks on target
All other areas	Bright dots within +/- 1mm of marks on target

3.4 Emitter Destressing

The aligned, collimated emitter is thermally destressed. The emitter will then checked by the quality control department for collimation and angular deviation (see following section).

- Secure 3 collimated, aligned emitters onto an emitter base plate.
- Place emitter assembly into thermal chamber and run the standard destressing profile (hydra.prf).

3.5 Final Alignment and Collimation Verification

This section is to be carried out after thermal destressing.
The aligned emitter is checked for collimation and for angular deviation by quality control.

NOTE: No fixture currently exists for checking collimation and alignment with 3 emitters mounted on a baseplate, so until there is another such fixture, the individual emitters will be checked on the current fixture.
Time required for this check is approximately 2min. per emitter.

Angular Deviation Check

- With the emitter bolted on the emitter alignment fixture, ensure the target is placed at 30" range. The laser line should be centred at an elevation of approximately 71mm +/- 1mm on the target (highlighted pink line indicated the correct elevation).

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

- If necessary, adjust the collimated laser line elevation with the adjuster screw.

Collimation Verification

- Place the HRPV alignment target at approximately 30" from the emitter base.
- Place the collimation verification target (strip with ~4mm hole spacings) 12" in front of the alignment target so that the laser line projects through the holes.
- While observing the pattern of the projected holes (pattern of bright dots) at the alignment target, adjust the verification target (or the alignment target) side to side until the bright dots are superimposed on the corresponding vertical dashes of the alignment target.
- Check the pattern of the bright holes along the length of the laser line against the marks on the alignment target and ensure the following:

Position Along Collimated Laser Line	Specification
Centre	Bright dots within +/- 0.25mm of marks on target
Ends (+/- 1")	Bright dots diverge (0.5 to 1.0mm) against marks on target
All other areas	Bright dots within +/- 1mm of marks on target

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1. OBJECTIVE

The laser line generator is to be assembled according to the required specifications.

2. INITIAL REQUIREMENTS AND PREPARATIONS

Required Parts/Tools

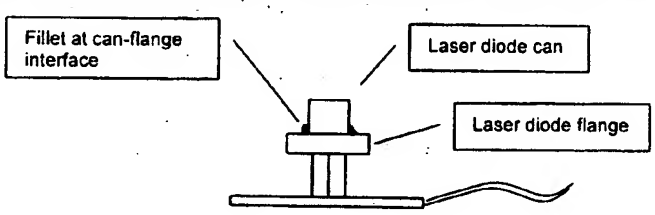
- Laser Line Generator Assembly Kit containing:
  Fully tested Laser Diode and Laser driver PCB assembly (3DM part # 11579, 11585)
  Kodak Line Generator Lens (3DM part # 11455)
  Laser Mount (3DM part # 1855)
  Retaining screw and standoff (3DM part # 11533, 11532)
- Ball-end allen wrench 5/64
- Power supply (5V, 150mA min.)
- Loctite E-20NS (3DM part # 11208) and dispensing system with pink nozzle tip
- UV curing epoxy (3DM part # 11043)
- Finger cots
- Reagent grade Isopropanol
- UV Curing Lamp
- Line generator Assembly/Alignment Fixture
- CMM

3. GENERAL PROCEDURE

3.1 Preparations and Laser Diode Alignment

Finger cots are to be worn, and 3DM optics handling and cleaning procedures are to be followed as appropriate. The time required for this stage is 3-6min per laser, and 4 hour cure time.

- Disassemble the Laser and PCB from the laser mount if required. Keep components together, since they are matched as an assembly.
- Clean and degrease the laser mount and laser diode flange with Isopropanol
- 'dry fit' the laser into the mount to ensure the diode seats fully into the mount.
- Apply a thin layer of epoxy into the laser mount counterbore and spread the epoxy completely around the counterbore perimeter. Apply a thin fillet (0.5mm – 1.0mm) at the base of the laser diode can-flange interface (see below), and a thin layer of epoxy around the perimeter of the laser diode flange.



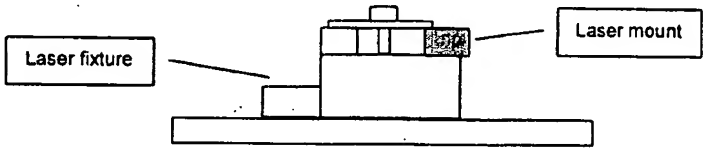
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

- Seat the laser diode into the mount with a turning motion, and secure the PCB with the washer and screw.
- Check the laser alignment: ensure the diode is fully seated, centred in the mount, and is parallel to the diode mount. Also check that the diode is not angled – gold mount inside diode window is vertical. Re-adjust if necessary.
- As an alignment check, power on the laser diode and check the elliptical spot has its long axis approximately parallel to the long axis of the mount (within +/- 3 degrees). Re-adjust by loosening the PCB and adjust with a twisting motion.
- The E20-NS will fixture within approximately 20 minutes at a temperature of 25C. To speed curing, place an incandescent lamp near the laser mount, but do not power the laser if heat is being applied.

3.2 Mounting and Aligning the Line Generator Lens

After the laser diode is secured in the laser mount and the epoxy is cured, the line generator lens is aligned and mounted. Finger cots are to be worn to keep the lens clean during handling. The time required for this stage is approximately 10-15minutes for alignment and fixturing – full cure time is extra.

- Clean and degrease the face of the laser mount with Isoproanol.
- Check the line generator for any dust or loose dirt – blow off dust or dirt. If the lens is dirty, obtain another clean lens.
- Mount the line generator fixture onto the CMM Z-axis probe.
- Clamp the line generator lens into the line generator mounting fixture, with the flat face of the lens centred against the soft tape strips.
- Clamp the laser mount with its front surface parallel to the front of the fixture and overhanging approximately 2mm. Place the laser alignment fixture onto the CMM (see below).



- Adjust the height of the CMM Z-axis probe until the centre of the line generator lens is at approximately the same height as the centre of the laser diode mount.
- Bring the fixtured laser mount up to the line generator lens and seat it flush against the lens flange.
- Power on the laser, and adjust the X, Y, Z axes of the CMM probe until the laser line is within the following specifications:
  1. boresighting is within specification (+/-2mm at 1000mm),
  2. the laser line is clean and free of any ghost lines between 100mm and 1000mm
  3. the laser line is straight, and horizontal (+/-1mm at 1000mm)
  4. the laser line intensity is symmetrical and as uniform as possible along the length and across the width of the line.
  5. the laser line is 6mm +/- 1mm thick at a range of 1000mm
- Tack the lens in place with UV curing epoxy, a small drop on each of the two top corners.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

- Once fixtured, release the lens by rotating the holding clips on the line generating fixture by 90 degrees.
- Pull the laser fixture away from the Z-axis probe, and secure the lens completely with UV curing cement around the lens, using caution to avoid any leakage of adhesive to the inside of the lens.
- Ensure that there are no gaps between the lens and the laser mount so that no debris can fall into the laser.
- Allow to cure fully under UV lamp.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

1820	3DM Devices - Technical Note	Jun 13, 2000
	LASER LINE GENERATOR EVALUATIONS	2

1.0 Summary

There is a requirement for a low cost laser line generator in 3DM products, in particular for the YETI, which has used expensive collimated diode laser modules with a 3DM line generator add-on and housing. Several line generator lenses and laser diode modules were evaluated and their profiles and efficiency measured to aid in designing a 3DM YETI laser.

2.0 Test Setup

The test setup consisted of the the following:

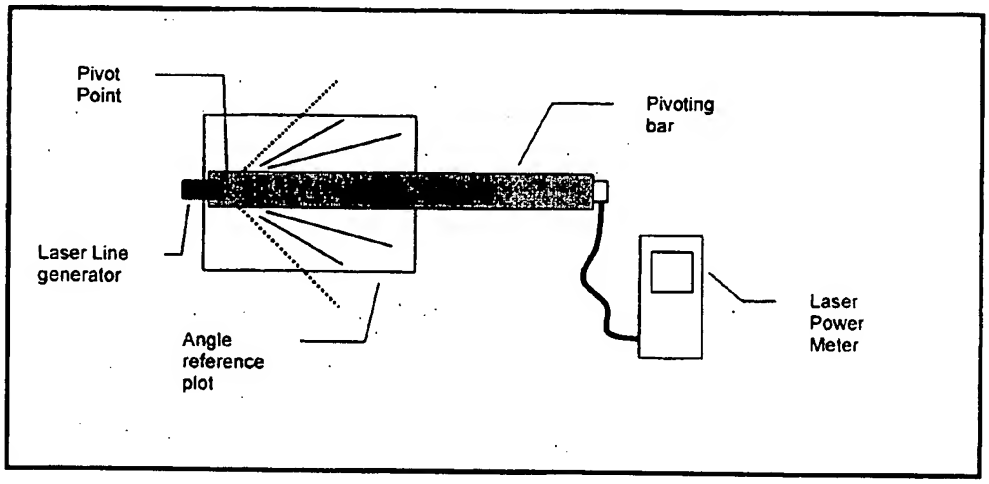
Line Generator under test,

Pivoting bar

Laser Power meter attached at the non-pivot end of the pivoting bar

Reference plot indicating placement for the pivoting bar at angles –45 to +45 degrees at 5 degree increments

The general arrangement is shown below. Note that the pivoting bar is rotated while the laser line generator remains in a fixed position.



Several different line generator types were evaluated in order to establish the true performance of each type. .In particular, the 'Uniform Intensity Line Generating Optics' as used in Lasiris products and the economical Kodak LG-P9 lens were evaluated.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

3.0 Discussion/Results

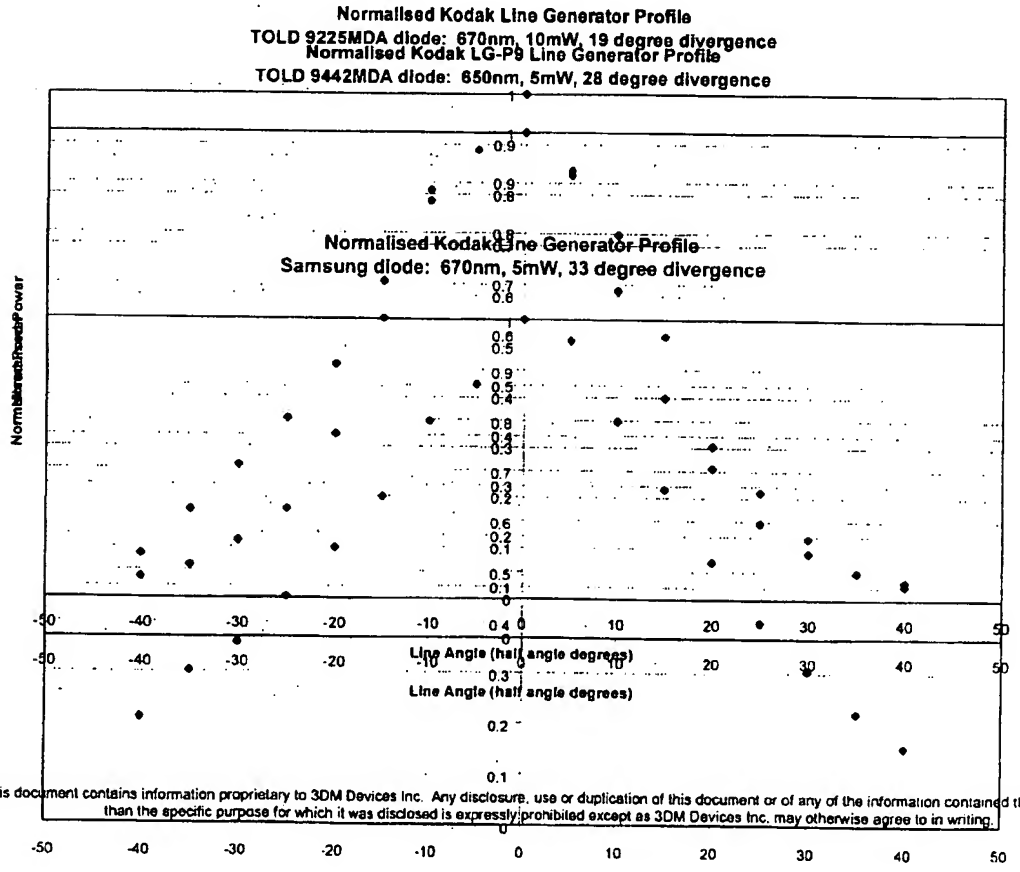
3.1 Laser Line Profiles

The discussion begins with intensity profiles for various line generators. In subsequent sections there will be presented a summary of the various line generators, comparing measured intensities and efficiency to help design a 3DM laser line generator for YETI.

3.1.1 Kodak LG-P9

The LG-P9 manufactured by Kodak combines a 9.5mm focal length collimator and 45 degree line generator into a single polycarbonate lens. The lens is very cost-effective and is simple to align to a laser diode. The resulting laser line generator has a typical Gaussian intensity profile similar to modules which employ cylindrical or 'rod' lenses. It is important to note that this lens is designed for lasers with a fast-axis divergence of 28 degrees, i.e. a 28 degree fast-axis divergence gives the 45 degree line (half-power beam width or HPBW). Thus, a lower divergence will not give a 45 degree line. This relationship has been investigated in these tests as well.

The measured intensity profile for the LG-P9 line generator coupled with a 28 degree fast-axis divergence diode is shown below. The line is approximately 45 degrees, and the distribution is indeed Gaussian in shape. Similar plots for different diode divergences are also presented.



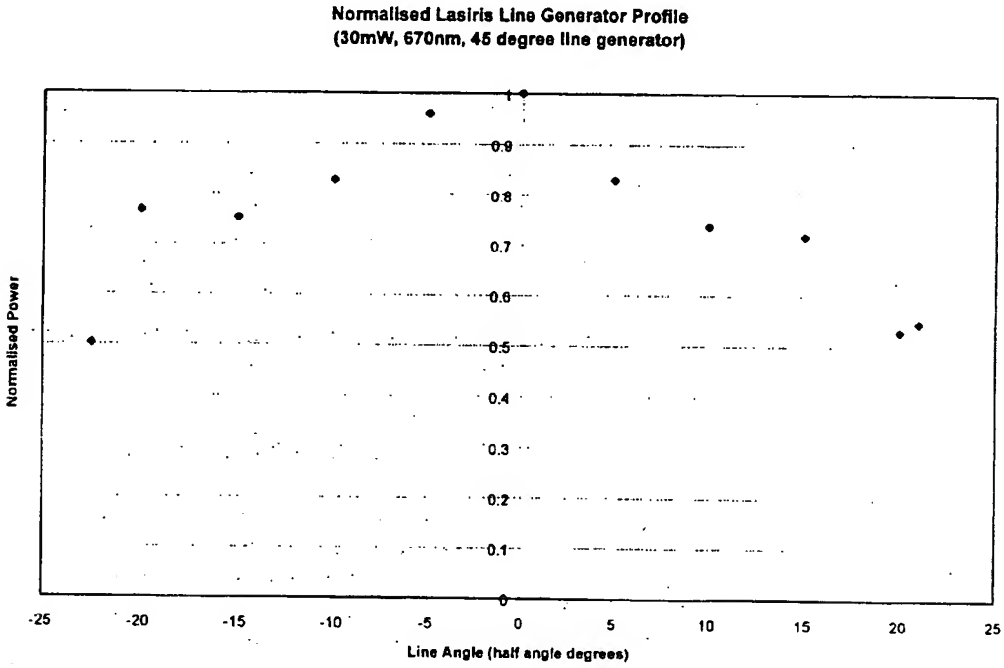
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The table below summarises the effect of laser line angle dependence on diode divergence.

Laser Diode Type	Fast-Axis Divergence	LG-P9 Fan Angle (HPBW)
Samsung SLD67018250, 670nm, 5mW	33 degrees	50 degrees
TOLD 9442MDA, 650nm, 5mW	28 degrees	45 degrees
TOLD 9225 MDA, 670nm, 10mW	19 degrees	29 degrees

3.1.2 Lasiris 45 Degree Line Generator

The Lasiris line generator is a patented 'uniform intensity' line generator which supposedly maintains an approximately constant intensity along the length of the laser line. The Lasiris documentation shows typical intensity variations of <25% along the main portion of the line (the ends of the line appear to have approximately 45% greater intensity than the centre of the line). The measured intensity of a typical Lasiris module (Hydra laser – 670nm, 21mW) is shown below.

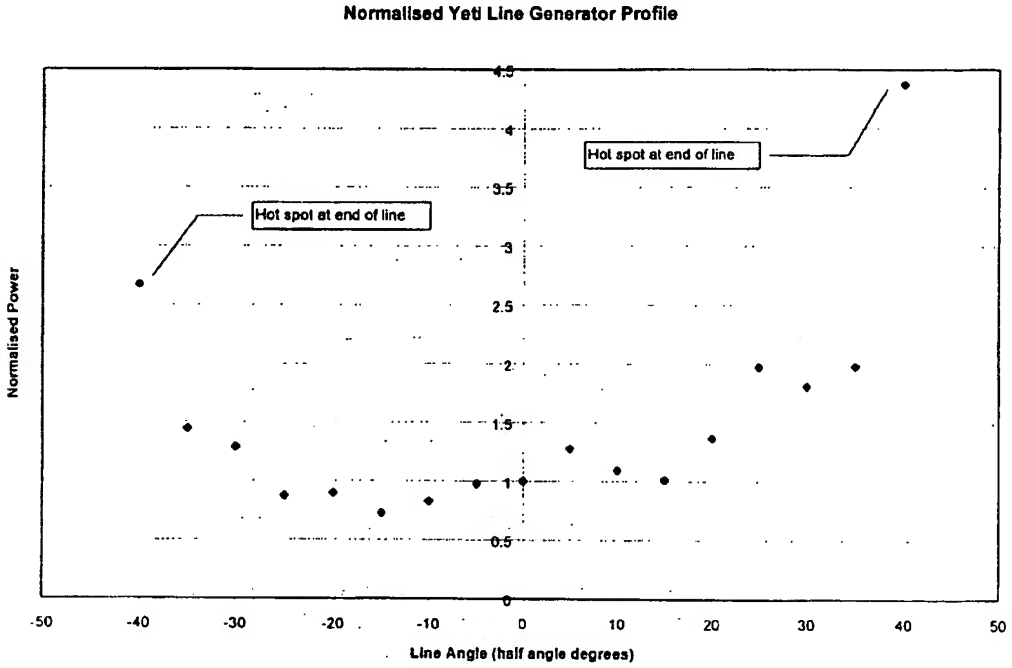


This profile differs significantly from Lasiris' advertised performance. This is not surprising given that the line generators are manufactured essentially by trial and error, whereby a line generator is deemed acceptable based upon a visual inspection of the line during the polishing stage.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

3.1.3 YETI Laser (R. Mathews M80 Lens)

The M80 lens is similar in design to the Lasiris lens, with one exception being that the M80 has an 80 degree fan angle. This lens is also a 'uniform intensity' line generator. A plot of the laser line profile for a typical YETI line generator employing the M80 is shown below.



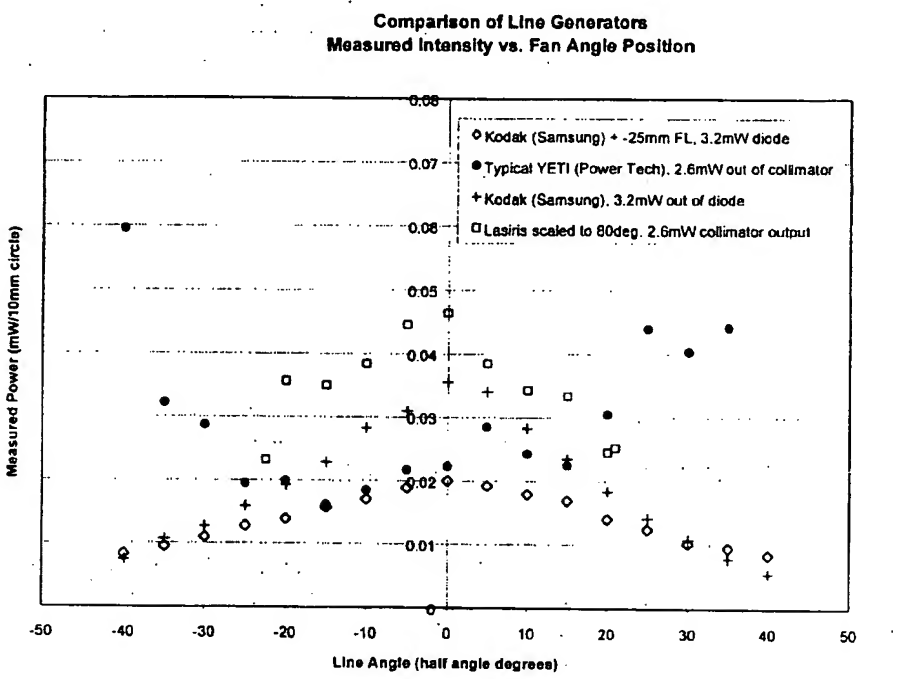
This profile, as with the Lasiris laser, is not quite 'uniform intensity'. Again, this is not surprising given that the line generators are manufactured essentially by trial and error, whereby a line generator is deemed acceptable based upon a visual inspection of the line during the polishing stage. The end points of the line are much more intense than the ideal (advertised) Lasiris. This is due to the overfilling of the line generator aperture, which is actually built for a 0.8mm-1.0mm HeNe laser beam. Note that it may actually be desirable to have more intensity at the end of the line depending on the object being illuminated by the laser line and the imaging geometry.

3.2 Laser Efficiency

There is a wide variation in efficiency for the different line generator types. The plot below gives measured intensities for several of the line generators for the 5 degree incremental angular positions. The measured laser power out of the diode or collimator is shown for comparison. Note that in addition to the previously evaluated line generator types, there is also included an 'expanded Kodak line generator'. This consists of the Kodak line generator, a highly divergent

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Samsung diode (33 degrees), and a negative cylindrical lens (-25mm focal length), and was an attempt to achieve an 80 degree fan angle using the inexpensive Kodak lens. This module is not practical due to several factors, the main factor being poor efficiency and the requirement of a highly divergent diode, which is not readily available in higher power diodes that are necessary to compensate for the poor efficiency.



3.3 3DM Laser Line Generator

The previous sections can be summarised in their relation to the YETI laser requirement as follows:

Mathews M80	Present YETI design uses the M80. The uniformity of a new 3DM Line Generator should be similar to that achieved with the M80.
Kodak LG-P9	Line profile is not adequately uniform for 80 degree line generator.
Expanded LG-P9	Adding negative cylinder improves line profile for 80 degree fan angle, but efficiency is poor and intensity too low. Increasing laser diode power is not straightforward since the required high divergence is not available for candidate high power diodes.

Therefore, the new YETI laser line generator must use the M80 lens. In order for the laser line to be uniform and adequately intense, the diode and collimator must be chosen appropriately.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

3.3.1 Collimator/Diode Considerations

The typical YETI diode modules which have been used with the M80 lens comprised of a 670nm 5mW diode and a 2mm collimator lens with 0.5 numerical aperture. Output power has been nominally 3mW. The table below gives approximate calculated spot sizes for the typical YETI module and some new proposed diode/collimator combinations. Note that 3 values are given for each collimator/diode combination based on the diode data: minimum, typical, and maximum.

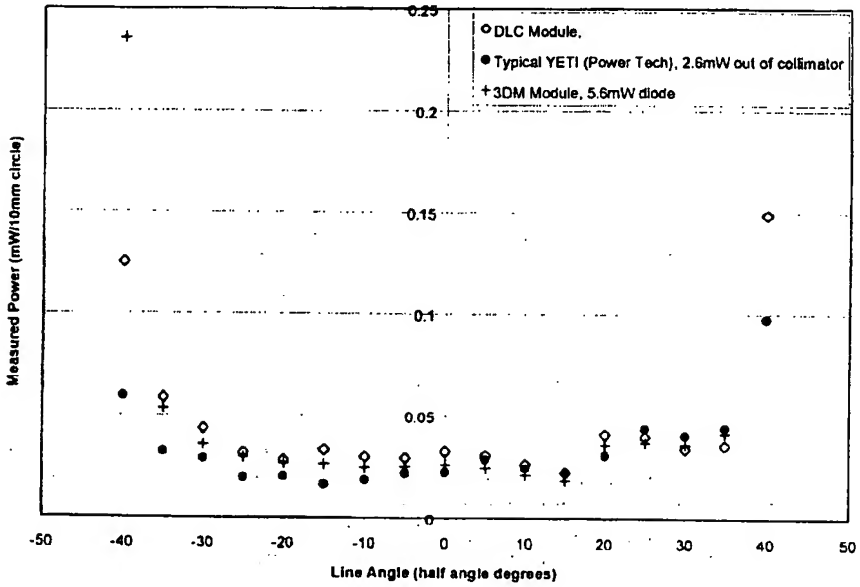
Module Description	Diode Description	Diode Divergence (deg)		Lens Description		Spot Size at face (mm)	
		Minor axis	Major Axis	Focal length	N.A.	Minor Axis	Major Axis
Typical Yeti diode laser module	670nm, 5mW unknown	5	22	2 mm	0.5	0.3	1.28
		8	30			0.47	1.72
		11	40			0.65	2.00
New Proposed Samsung laser with philips AC256T lens	670nm, 5mW Samsung	7	22	4.35mm	0.5	0.9	2.79
		8	33			1.03	4.09
		10	38			1.29	4.40
New Proposed Toshiba, 10mW laser with philips AC356T lens	670nm, 10mW Toshiba	5	15	4.35	0.5	0.64	1.92
		8	18			1.03	2.30
		11	23			1.41	2.91

The typical Yeti diode module fills the M80 line generator quite efficiently, with a slight rotation of the elliptical diode spot - usually approximately 20 to 30 degrees, such that the major axis is 20-30 degrees from vertical, and the laser line is horizontal. This 'spreads' the minor axis while the collimator is focusing the major axis of the elliptical spot. This results in a line which converges sharply at the focus (approx. 8") and has a short depth of focus (44mm approx).

The 10mW Toshiba diode (currently being used in Scangogh) offers a low divergence, which produces a relatively small spot size at the module face. The additional power available can be used to compensate for some of the expected efficiency problems due to any overfilling of the line generator lens when the spot/line generator orientation is such that the major axis is spread by the line, and the minor axis is being focused. This will provide a more consistent line thickness due to the larger depth of focus. This scenario is proposed for the 3DM YETI line generator. A comparison of this proposed line generator with the typical YETI laser follows.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

YETI 80 Deg. Laser Line: Comparison of Typical, DLC, 3DM
(3DM: TOLD9225MDA diode 5.58mW Diode Output, AC258T collimator)



In the above plot, there is an additional laser for comparison noted as 'DLC Module'. This is a low-cost collimated laser which was employed in YETI #P16, and was aligned such that the minor axis was spread and the major axis was focused down. With such an alignment, this module had a larger spot diameter at the module face (3mm approx.), but produced a satisfactory line profile in the scanner's working range, although the line was not as fine as with the 'typical' modules.

The above plot shows that in terms of measured intensity profile, the proposed 3DM YETI laser is very close to the typical module. The only problems are the intense endpoints, which will be reduced by an aperture at the line generator lens face. A significant benefit of the proposed laser is the improved depth of focus of the line.

4.0 Conclusions

A replacement laser line generator for the YETI is proposed, using the current M80 line generator, a Toshiba TOLD9225MDA diode (currently used in Scangogh), and an economical Philips collimator lens. This line generator gives similar line intensity profile with improved laser line depth of focus, and will provide significant cost savings with appropriate fixtures and packaging.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

3DM
Devices Inc

HYDRA - Hub

SYSTEM DESIGN DESCRIPTION

Sept 2000 Rev 1.5

Prepared for:

3DM Devices

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Revision History:

Version #	Date	Author	Description
0.0	Oct 14, 1999	JK	Concept Release - For Discussion
1.0	Nov 12, 1999	JK	Reconfigured as multipurpose sensor i/f Hub
1.1	Jan 10, 2000	JK	Command structure, H/W details added
1.2	Feb 06, 2000	JK	To MPC8260, Cmnd descriptions 50% complete
1.3	May 21, 2000	AC/JK	Cmnd descriptions complete, Hotlink details added
1.4	Aug 21, 2000	AC/JK/ML	Revised command format, detailed operation
1.5	Sep **, 2000	AC/JK/ML	Added FP_SEGS structure Revised Hub I/O pin assignments

Referenced Documents:

Document	Source
MPC8260P PowerQUICC II Technical Summary	Motorola MPC8260/D
EST Single Board SBC8260 Eval System	http://www.estc.com
HOTLink Transmitter/Receiver - CY7C924DX.pdf	Cypress Semiconductor

Print Date: February 1, 2001 (9:36AM)

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

TABLE OF CONTENTS

Purpose **4**

Scope **4**

Intended Audience **4**

1.0 Overview **5**

1.1 Features 6

1.2 Specifications 8

2.0 Architecture **9**

2.1 Overview 9

2.2 Hardware 9

2.2.1 Memory Map 10

Hub I/O 10

2.3 Logical 11

2.3.1 On/Off Line 11

2.3.2 Hub Services 11

2.3.2.1 Hub Control 12

2.3.2.2 Frame Processor 12

2.3.2.3 Sensor Control 12

2.3.3 Sockets 13

3.0 HUB I/O Subsystem **15**

3.1 Overview 15

3.2 Protocol - Direct Connection 17

3.2.1 Online 17

3.2.2 Offline 18

3.3 Protocol - Multi-drop Connection 19

3.3.1 Online 19

3.3.2 Off Line 20

3.3.3 Loop operation 20

3.3.4 Enumeration 21

3.4 Hub –> Sensor Command Set 23

3.5 Sensor I/F Operation 25

3.51 Sync 25

3.5.3 Power up sequence 25

3.5.4 Reframe 26

3.5.5 Commands 27

3.5.6 Endian and Address Issues 27

3.5.7 Device Selection and Register Decode (little endian) 28

3.5.8 Channel Data Transfer to Memory 28

DMA Transfers 29

Interrupt Transfers 29

3.5.9 MPC to/from FPGA Bus Transfers 29

3.5.11 State Machines 31

3.5.12 CONTROL REGISTER (little endian) 32

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

3.5.13 COMMAND REGISTER (effectively two registers here controlled by expand bit) 32
3.5.14 FPGA Status Register 34
3.5.15 Channel (3..0) Status Registers 34
3.5.16 Sync Registers [3..0] 34
3.5.17 Load Transmit Data 34
3.5.18 Serial Address Register Access 35
3.5.19 TXSTOP# and data transmission 35
3.5.20 Unused and unsupported pins 35
3.5.21 Fanned out pins 36
3.5.22 Interrupts 36
3.5.23 Encoder 36
3.5.24 Output Data Bus 36
3.5.25 BIST 36
3.5.26 Data Throttle - Sync Pause 36
3.5.27 Data Packet 37

4.0 Hub Firmware - VxWorks 39

4.1 Overview 39

4.2 Hub Control Tasks 39
4.2.1 Hardware boot and self test 39
4.2.2 VxWorks boot 40
4.2.3 Hub Start Up – Application entry point 40
4.2.4 hubIO Task 40

4.3 Sensor Tasks 41

4.4 Frame Processor Tasks 41

4.5 Console Tasks 41

4.6 Error Handling 41

4.7 Diagnostics 41

5.0 Hub Host Commands 42

5.1 Overview 42
5.1.1 Message Format 42
5.1.2 Command Format 43
5.1.3 Response Format 43
5.1.4 Data Format and Sizes 43

5.2 Hub Socket Commands 44

5.3 Sensor Socket Commands 48

5.4 FP Socket Commands - HRPV 49

6.0 Sensor Configurations / Performance 54

6.1 HRPV 54

6.2 High Speed Camera 55

6.3 Hydra Heads 55
6.3.1 General 55
6.3.2 Data Rates 55
6.3.2.1 Profiling 55
6.3.2.2 Video 56
6.3.3 Modifications Required 56

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

6.3.3.1 Hardware 56
6.3.3.2 Firmware 57
6.4 Transverse Frame Profiling – LPS2 57
Appendix I – MPC8260 VxWorks Boot Configuration 58
EST SBC8260 Boot 58
Configuration 58
Boot Sequence 58
Downloaded VxWorks Image 61

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Purpose

This document is the design reference for the Hydra communications and data processing hub.

Scope

This document provides a design description of a standalone communications and data processing hub. The description encompasses the hub hardware and software, the Host and sensor(s) interface requirements, and the system performance in a number of scenarios.

The Hub was originally intended to replace the VME Hotlink card and associated processor card in a Hydra scanner system. In the design evolution process, several other applications of the Hub have become apparent and it is now planned as a general purpose sensor controller and data gatherer. The scope of this document has expanded to address these other applications and the specific Hydra requirement is now just one of these.

Intended Audience

This document is intended for internal review at 3DM Devices and comment from outside parties.

All use of this document is subject to the terms of non-disclosure agreements.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

1.0 Overview

This document describes the design of a standalone **Hub** that provides a platform independent control and data interface to a variety of high speed optical sensors. This development is driven by the need for a more cost effective and flexible system that is independent of the system backplane (VME, PCI etc) and Host operating system. The Hub is intended for use in measurement systems where data from a variety of sensors must be collected, time stamped with an encoder value, processed and assembled into real world values and then transmitted to a Host system on demand.

The terms head and sensor are used interchangeably in this document.

The Hub design has been developed with the following sensors in mind –

Existing Hydra heads

High Resolution Plan View – Laser Caliper

High Speed CMOS camera system

Existing LPS ranging sensors as used by CAE Newnes in transverse frames

Others tbd

The Hub is a PowerPC based device that accepts data at high speed from multiple sensors, assembles this into calibrated, time stamped, real world measurements, and transmits this as requested in a timely manner to a Host system. This approach places all sensor specific tasks such as head communication, profile assembly, calibration and head initialisation in a standalone hub which connects to the Host system via a standard Fast Ethernet connection.

The key considerations in selecting this system topology were as follows:

The hub provides a Host independent control and data processing platform. Sensors can be deployed in different applications and in various mixes with minimal or no re-engineering of the interfaces for different Hosts.

The hub supports custom physical and logical connections to the heads that are optimised for speed and cost and provide the head synchronisation signals.

The Hub accepts a direct encoder input so that all data can be accurately time stamped at point of collection.

The Fast Ethernet Host connection provides a cost effective and standard protocol that continues to gain acceptance as a manufacturing floor communication standard. There is good availability of both copper and fibre based network components. The point to point nature of this implementation largely negates the non-deterministic nature of the interface, although this may be irrelevant anyway given that the data is time stamped.

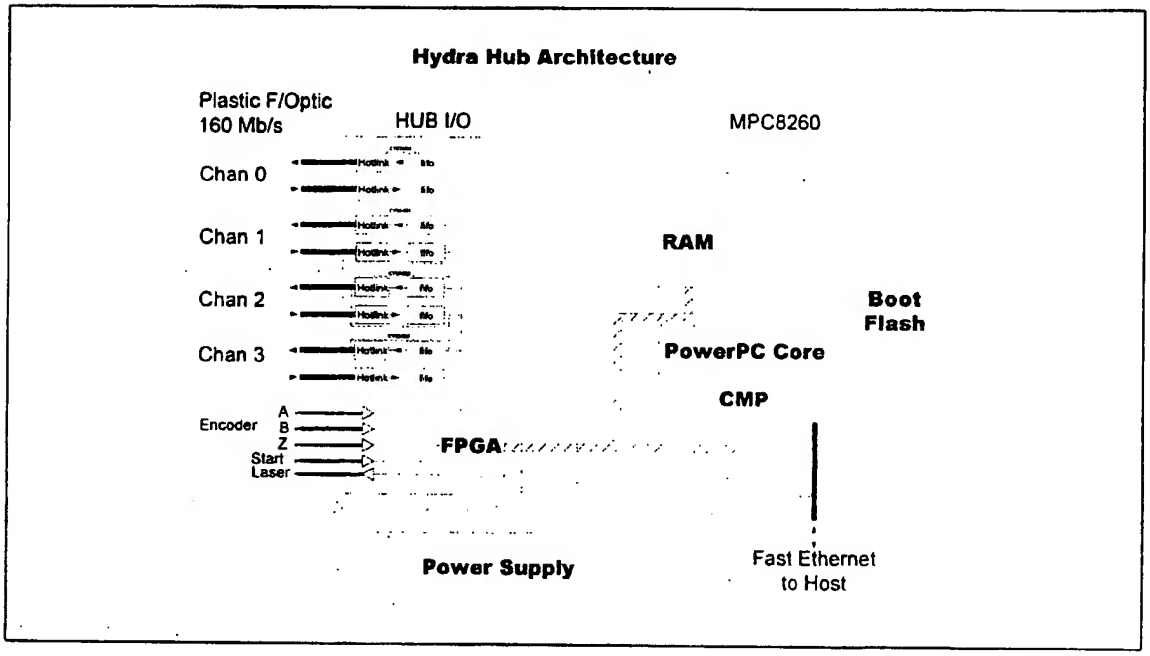
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

All hub signal connections can be implemented in fibre to minimise emissions and susceptibility concerns.

Sensor system boot and calibration table loading can be handled at the sub system level by the Hub with no Host intervention.

The new system configuration is cost effective and versatile as compared to the existing Hotlink interface and VME based processor system.

1.1 Features



Features of the hub design are as follows:

Controller/Processor

MPC8260 family PowerQUICC II processor with 32bit EC603e PowerPC core and Communications Processor Module (CPM) supporting 10/100 base-T with media independent 100 Mbps interface.

The MPC8260 features two separate external data busses – 60x and Local – to maximise independent operation of the CPM and PowerPC core. On chip dual port RAM provides an efficient data and control path between the two processors. The Hub design utilises the local bus for Hotlink and Fast Ethernet communications, with the 60x bus reserved for the PowerPC core computational functions.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

VxWorks operating system

Flash resident boot image.

On board LUT's for calibration data. LUT's are loaded on boot from heads transparently from Host.

Head Interface

4 channel duplex interface, memory mapped for fast DMA transfers

Cypress CY7C924DX integrated Tx/Rx Hotlink controllers with on-board FIFOs using plastic optical fibre (POF) for the physical connection.

Supports single and multiple daisy chained heads per channel.

Host Interface

100 Mbps Media Independent controller. Interface implemented with fibre optic option as default

Protocol provides calibrated profile data from Hub.

Miscellaneous I/O

All I/O is implemented with plastic fibre optic cable. The I/O is assigned to the following logical connections

Inputs -	EncA	Encoder Quadrature Channel A
	EncB	Encoder Quadrature Channel B
	EncZ	Encoder Index Z
	Start	Board Start Pulse
Output -	Laser Control	

Package

Standalone, fully enclosed metal box

110/220 VAC input

LED diagnostic and status indicators

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1.2 Specifications

Inputs

4 channels – Hotlink – plastic fibre – 160 Mbaud – max 50 metres

4 fibre optic general purpose – 1 Mbaud - max 50 metres

Outputs

Fast Ethernet – optical fibre or 100BASE-TX

1 fibre optic general purpose

Front panel diagnostic LEDs

Power

110/220 VAC universal input

Processor Subsystem

MPC8260 family PowerQUICC II processor

66 MHz system clock

32 MB SRAM

8 MB Flash

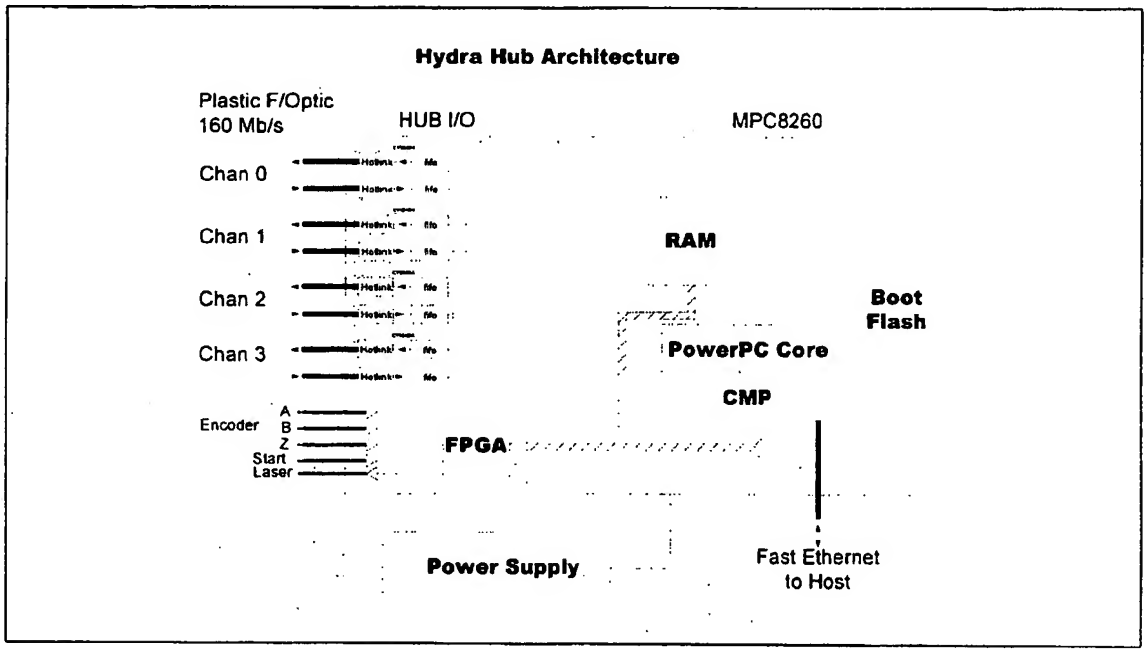
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibiled except as 3DM Devices Inc. may otherwise agree to in writng.

2.0 Architecture

2.1 Overview

The development and pre-production hardware design is based on a commercially available MPC8260 based single board computer with the Sensor Interface implemented in a plug in daughter board. Full scale production hardware will be a custom pcb based on the same architecture with the Sensor interface either on-board or as a daughter board.

The Motorola MPC8260PowerQUICC II Quad Integrated Communications Controller (PowerQUICC) is a one chip integrated microprocessor and peripheral combination intended for a variety of controller applications. The CPU is a 32 bit PowerPC implementation that supports integer (floating point tbd) operations on a 32-bit internal data path and 32-bit arithmetic hardware. Externally, the MPC8260 supports both a 64 bit 60x style bus and a 32 bit local bus. These two busses facilitate independent operation of the CPM and PowerPC core.



The operating system is VxWorks booting directly from Flash memory on the single board computer.

The system is designed to boot autonomously and configure attached sensors to stored configuration data if so configured from the previous power down.

2.2 Hardware

The main development components are

SBC8260 Single Board Computer (EST Corp)

MPC8260 Processor operating at 66 MHz

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

16 MB DRAM
8 MB Flash
RS-232 Monitor port
Ethernet 10Base-T on board
Fast Ethernet 100 Base-T MII on-board

Hub I/O plug in card

4 Cypress Cy7C924DX Hotlink Transceivers
4 HP POF Receivers
4 HP POF Transmitters
1 Altera FPGA
4 HP POF Receivers
1 HP POF Transmitter

Power Supply

Condor Universal Input

The Hub I/O card is mounted above and parallel to the main board and connects to the Local Bus on the SBC8260 using a PCI style edge connector.

2.2.1 Memory Map

The memory map of the SBC8260 is shown below.

Chip Select	Function	Start Address	End Address	Assigned to
CS0	32 bit General Purpose	FE00 0000	FFFF FFFF	Flash – debug monitor – JP24 selected
CS1	32 bit General Purpose	FC00 0000	FCFF FFFF	Expandable Boot Flash – not installed
CS2	64-bit Standard SDRAM	0000 0000	00FF FFFF	603 Bus RAM
CS3	64-bit expandable SDRAM	0100 0000	01FF FFFF	603 Bus Expansion RAM – not installed
CS4	32-bit SDRAM	0400 0000	04FF FFFF	Local Bus RAM
CS5	8-bit General Purpose	2200 0000	2200 FFFF	8K EEPROM
CS6	8-bit General Purpose	E000 0000	E0FF FFFF	Flash – VxWorks boot image – JP24
CS7	8-bit General Purpose	2100 0000	2100 FFFF	Controls/PALs/LEDs
CS8	**UPMC**	2300 0000	2300 FFFF	**Hub I/O**
CS9	64-bit UPMA	User Spec	User Spec	
CS10	64-bit UPMB	User Spec	User Spec	
CS11	64-bit UPMC	User Spec	User Spec	

See section 3.5.6 Endian and Address issues for more discussion of memory addressing.

The configuration of the UPMC memory access state machine and assignments of the address spaces into VxWorks is done during system boot using modified routines in the following modules which are required to build a VxWorks boot image:

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

romInit.s	*memcUpmCInit*	(sets up UPMC state machine – assembler code)
syslib.c	*structure sysPhysMemDesc*	(describes physical memory to VxWorks)
config.h	*defines for HUB IO Base Address*	

2.3 Logical

The logical design of the hub provides the foundation for interfacing a variety of different sensors, now and in the future, with minimal changes at the low level hardware and software driver interface level. There is a common format for communication between the Hub and all sensors, and a standard partitioning of tasks between various subsystems to accommodate present and planned sensor systems. This arrangement provides a common basis for new and future sensor interfaces and is intended to preserve system integrity and compatibility, whilst retaining the benefits of custom high speed links to each sensor type wherever possible.

The design approach is similar to that of Win95 plug and play, the Universal Serial Bus, and Firewire interface standards in which a device (sensor) must adhere to a common interface protocol at the lowest level and essential differences are implemented at higher level by the application developer. It is then possible to quickly interface a new sensor type by adding only the high level interface software routines to access the lower level standard data i/o and control functions. All low level system interfacing is provided by Hub Control services operating independently of sensor type and command or data content - there is no re-coding of low level interrupt service routines to add new message types, new data formats, or additional sensors.

The Dynamic Configuration (Plug and Play) feature of the Hub is an integral part of support for multiple sensor types, standalone system boot, and the proliferation of smart sensors in which details of the sensor type, driver requirements and calibration are embedded in the sensor non-volatile memory at factory build. Each sensor must adhere to the hardware interface standard based on the Hotlink fibre optic implementation, and provide at least the minimum responses on start up to support mandatory sensor type identification and enumeration by the Hub Control routines to permit proper system configuration.

The hub provides a well defined platform for the application developer to implement specific data gathering and conversion routines to handle data from his/her sensors which are then grouped into an appropriate frame processor configuration accessible to the Host. The set up and control of the hotlink interfaces, encoder interface and system synchronisation is handled by the core hub control software. Services are provided as appropriate to each sensor application.

2.3.1 On/Off Line

The term ON/OFF LINE describes the two states of the complete system – it does not refer to connections to individual sensors or channels. It might more properly be described as acquiring and not acquiring data.

In offline mode, sensors accept commands and configuration data in a non-time critical fashion and are typically addressed individually. There are small amounts of data to be moved in either direction, except for diagnostic gray scale video. The Diagnostic Host is the predominant connection to the Hub.

In on-line mode, the sensors operate in framing mode in which data is acquired in continuous frames at a rate set by the system clock. Data is read continuously from each sensor, converted and assembled in the Frame Processor and passed to the DP on request. The DP is the predominant connection to the Hub.

2.3.2 Hub Services

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The logical architecture is separated into Hub control, Frame Processor and sensor service modules. The allocation of activities amongst these is outlined in point form below.

2.3.2.1 Hub Control

Start Up
- VxWorks boot
- Hub hardware initialisation
- Hotlink interface reframe
- Sensor polling and enumeration
- System configuration and sensor task initialisation
- Frame processor task initialisation

Off line
- Sensor I/O – DMA
- Hardware monitor
- Error handling

On line
- Sensor I/O – DMA
- System synchronisation signals
- Encoder readback

2.3.2.2 Frame Processor

Start Up
- Calibration table setup
- Data buffer allocation

On-line
- Data assembly and conversion
- Encoder time stamping
- Profile data output to Host

2.3.2.3 Sensor Control

Start Up
- Status Read
- Configuration of sensor
- Read of calibration coefficients

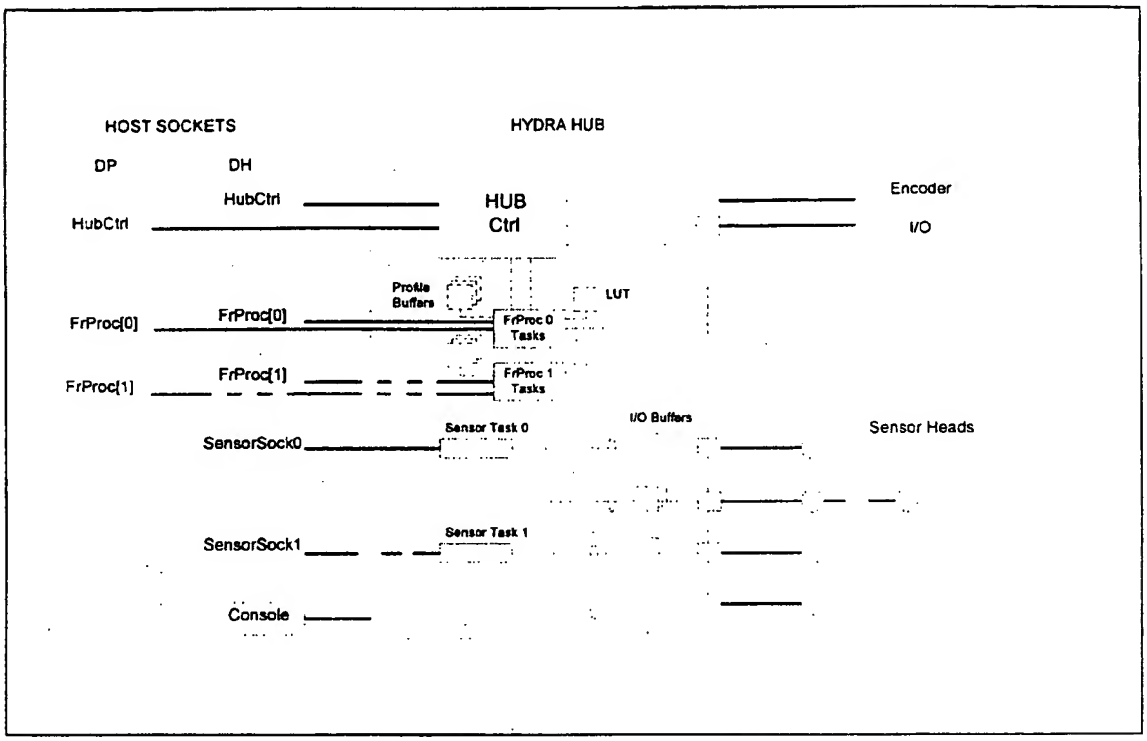
Off Line
- Configuration changes
- Diagnostic data tasks – video, raw data etc

On Line
- none

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

2.3.3 Sockets

These services are handled by groups of VxWorks tasks and are accessible to the Host through associated sockets as shown below:



'Host' is a generic term and is not defined as a single entity. It will typically comprise at least two functional blocks – a real time decision processor, or DP, providing control information to downstream systems, and a user interface application for system setup – the Diagnostic Host, or DH. Each of these may reside on the same hardware, or more often the DP will be an embedded processor in the same cabinet, and the DH application will be running on a networked computer.

The port numbers for HubCtrl and HubCtrlOnline are hardcoded, and these ports are always opened for connection after start-up. All other port numbers are retrieved with the appropriate command – eg. HH_GET_FP_IDENT to get fpIdent which contains the fpSktOffline and fpSktOnline port addresses, and the respective ports are not opened except in response to an appropriate LAUNCH command – eg HH_LAUNCH_FP.

The DP or other supervisory task connects to the online HubCtrlOnline port for Start Up, Hub Status and error recovery operations. The DP then connects to the online fpSktOnline port to request assembled data as necessary. The DH uses all the other sockets and an alternate Frame Processor socket for system configuration, diagnostics and monitoring. The DH is not required for run-time operation.

The HubCtrl socket provide access to all hardware on the hub. The online version of this socket provides the same functionality except with no access to the HH_SET_CHNL_CTRL, and HH_SET_CFG commands.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

One sensor socket is assigned per sensor type. The sensors may be spread across several channels and there may be more than one per channel. There may be more than one sensor type per system.

The Frame Processor socket pairs, (DP and DH), access the tasks responsible for gathering, converting and assembling the data from a group of sensors. There may be more than one pair of Frame Processor sockets depending on the mix of sensor type and location attached to each Hub.

In a typical application, the Hub will boot and configure itself from on-board Flash. The DP or supervisory task will establish connection with the HubCtrlSkt, confirm configuration data and then request opening of one or more online FP sockets, establish connection to these and then go online. There is no requirement for any other action from the DP or other system resource.

The Diagnostic Host on an NT platform is used for system setup as required.

The sensor sockets are used primarily for diagnostics. The hub socket is used for diagnostics, system software updates and configuration purposes, and system monitoring.

The Console socket is a debug connection which can be conveniently accessed via telnet. This is in addition to the standard vxWorks serial debug port to which printf output is usually directed.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

3.0 HUB I/O Subsystem

3.1 Overview

The Hub I/O is a plug-in pcb providing all i/o communications with attached sensors and encoder. There is provision in the design for ganging more than one Hub I/O card onto a single PowerPC, but the preferred means of system expansion is daisy chaining sensors of the same type onto a single channel and operating this in multi-drop mode.

The card provides four duplex Hotlink channels, memory mapped to facilitate DMA transfers, with an FPGA providing the Cypress to memory bus interface and Hotlink control logic.

The four channels can support a variety of sensors with these connected one per channel, or daisy chained with multiple sensors of one type per channel. The two modes of operating a channel are termed direct and multi-drop respectively.

The Hub I/O transmits a programmable sync pulse on every channel for sensor synchronisation to a system clock or external encoder. This sync pulse is an integral part of the Hotlink communication protocol with each sensor.

The following discussion assumes a generic sensor that must have a Cypress transceiver as the interface element, an FPGA (or equivalent) based state machine, and/or a microcontroller if extended syntax commands are to be supported.

The Hub/Sensor interface requirements can be summarised as:

Start-up and configuration in offline mode, followed by prolonged periods of on-line operation.

In offline mode, sensors accept commands and configuration data in a non-time critical fashion and are typically addressed sequentially with adequate time for handshaking etc. There are small amounts of data to be moved in either direction, except for diagnostic gray scale video.

In on-line mode, the sensors operate in framing mode in which data is acquired in continuous frames at a rate set by the system clock. This data must be read from each sensor in a timely fashion and there is typical no time and no requirement for checking and retransmission of corrupted data. Data flow is predominantly from the sensor to the Hub.

The Hub design makes use of several features of the Cypress transceivers to facilitate data transfers and off-load most of the control logic to FPGA and DMA routines.

In addition, a design aim is to fix the low level protocol so that new sensor types and command sets can be added without the need to modify or add code at the driver level to support these.

The Cypress design features of significance are:

Built-in receive and transmit FIFOs, 256 bytes each, with hardware FULL, 1/2FULL, and EMPTY flags.

Support for tagging data bytes with cell markers in the transmit FIFO's and hardware pacing of transmit activity by a simple transmit-next-cell instruction.

Seamless toggling of remote receiver RXINT flag in response to toggling of local transmitter TXINT line.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Support for multi-drop network architecture with built-in hardware address registers.

These features make possible a packet based protocol, using a 128 byte packet size for data transmission **from** the sensor. The cell marker is written at 128 byte intervals in the Transmit FIFO and permits hands-off transmission of the preceding 128 bytes by FPGA logic. Receipt of these at the Hub Receive FIFO triggers a 1/2FULL condition which raises a DMA request to transfer 128 bytes from the FIFO. Data flows continuously with a FULL condition (very unlikely) at the Receive FIFO causing a pause in the transmission from the sensor. This pause is signaled by the Hub using the RXINT toggle as described next.

The TXINT/RXINT toggle is provided by the Cypress logic as a seamless insertion of special characters to set RXINT HI or LO to mimic a change of state of TXINT. The primary use of this feature will be broadcast of the system time base to all sensors with no impact on data transfers in progress. As noted above, this feature will also be used to throttle data transfers from sensors **connected in direct** mode – ie one sensor per channel, as follows. The system time base signal is defined as a short pulse of fixed duration, with the falling edge of this defined as the actual signal. If the pulse extends for longer than the predefined duration, it is an indication of a FULL condition at the Hub receive FIFO and sensor data transmission is inhibited until it drops LO again. The RXINT signal may thus toggle at any point in time, the FPGA logic differentiating between timebase and RxFULL and acting accordingly. In the event that the RxFULL condition persists up to the next timebase slot, the FPGA logic abandons the current frame, resets the Transmit FIFO, and attempts to proceed with the next frame of data. This process is possible as the FPGA logic in the sensor maintains its own local clock in rough synchronisation with the system time base.

In summary, data transmission from a sensor in direct mode is driven by the sensor, with throttling provided by the Hub if necessary. This provides the fastest rate of data transfer.

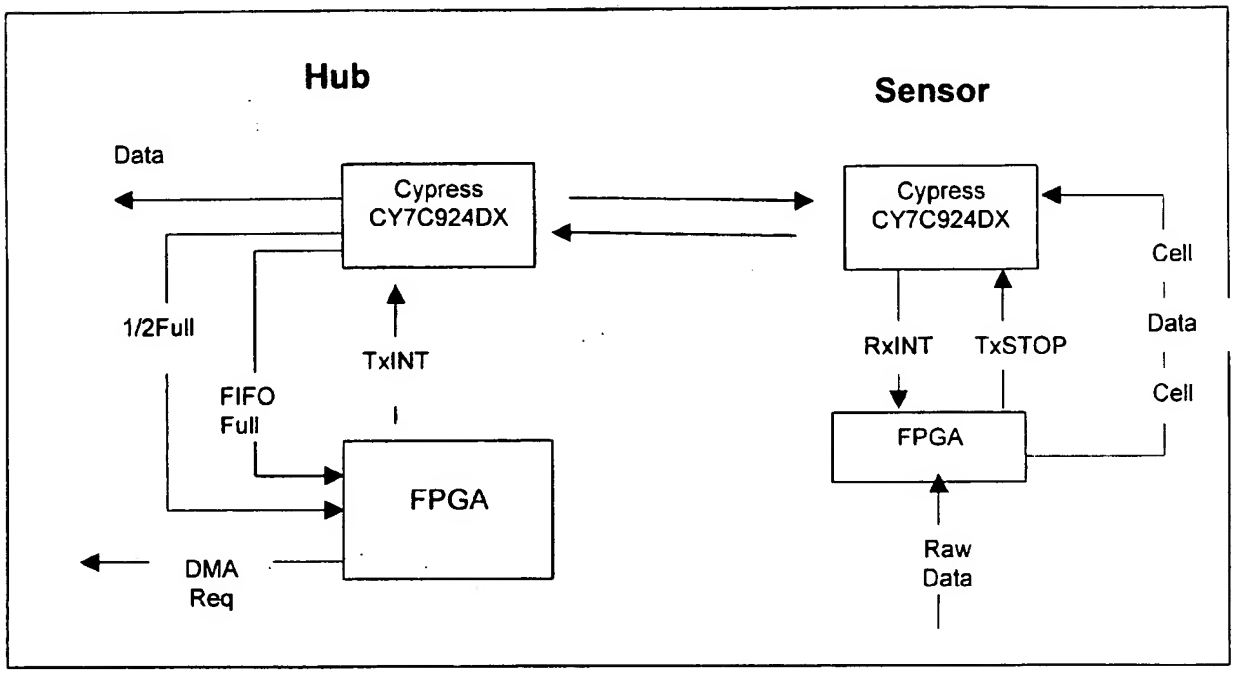
In contrast, in a **multidrop** connection, heads must be addressed individually in sequential fashion and the Hub polls each head to request a packet of data. This is a slower data transfer protocol, but should be compatible with the type of sensor that is daisy chained. The RxFULL condition signaling is not applicable as the Hub will not request data if the Receive FIFO is full. However, there must be a timeout condition imposed on the response to a poll by a head to avoid system stall.

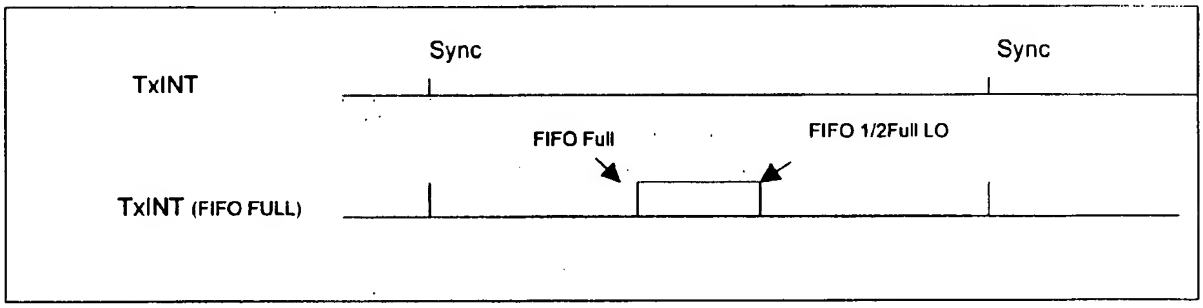
The discussion above focussed on the sensor to hub direction of data flow. The Hub -> sensor protocol is similar but accepts variable length packets as described in the detail discussion below for all data transfers.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

3.2 Protocol - Direct Connection

There is only one sensor connected to a channel as shown below.





3.2.1 Online

Hub -> Sensor

The Hub inserts system timebase sync pulses in the data stream by toggling TxINT for a defined interval. These are received as RxINT at the sensor. The FPGA interprets these as Sync pulses if of correct duration, otherwise these are an indicator that the Hub FIFO is full and data transmission should be paused.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

The Hub does not write any commands or data to the Transmit FIFO in on line mode except for the OFFLINE_CMD

Sensor -> Hub

The sensor writes data when available to its Transmit FIFO and tags the first character of the next 128 bytes with a Cell marker. The 128 bytes up to the Cell marker are transmitted automatically when TxSTOP is asserted momentarily. TxSTOP is asserted if there is not a pause condition from the Hub.

The Hub receives the 128 byte packet which sets the FIFO ½ full flag. The FPGA issues a DMA request to read the 128 bytes. If the DMA does not occur in a timely fashion, the FIFO Full flag will be set on receipt of the next 128 bytes from the sensor. This causes the FPGA to assert TxINT until the FIFO empty flag is asserted – ie the first DMA and second DMA have completed.

3.2.2 Offline

Hub -> Sensor

The Hub writes command strings to the Transmit FIFO with the last byte tagged with a Cell marker. The string is received by the Sensor and the Cell marker asserts the RXSOC – Receive Start of Cell marker. Note that the Hub transmitter is in continuous transmit mode – TXSTOP is deasserted – and the Cell marker is transmitted and is really an end of cell mark in this instance. The Sensor FPGA logic monitors RXSOC and uses this to determine when a complete command string has been received. The command string is read one byte at a time by either the FPGA or microcontroller.

Command strings may extend to 256 bytes

Sensor -> Hub

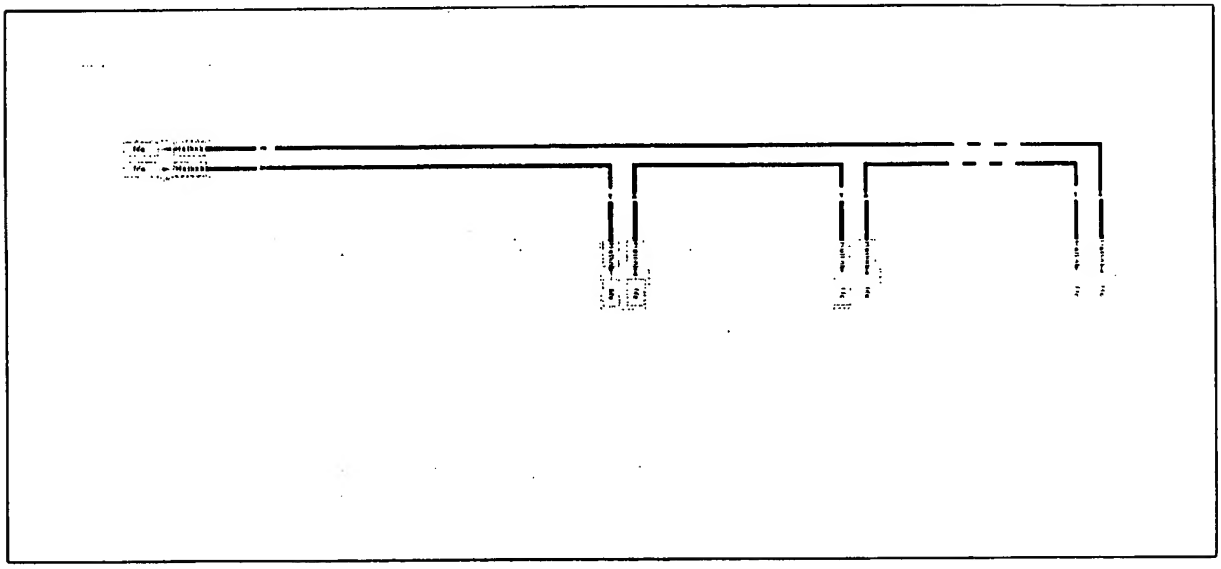
The sensor responds to commands with 128 byte packets starting with a Cell marker in exactly the same fashion as on-line mode. This obviously imposes extra overhead on short responses but minimises any problems with missed packets etc when switching to/from online mode. The Hub DMA routine always responds to a ½ full FIFO condition and transfers 128 bytes to memory, and so there should be no dropped bytes, timeouts etc as there could be with different length responses from the sensor. The higher level protocols are responsible for checking for an appropriate and timely response from the sensor.

Note that the highest data load in offline mode is gray scale video, and so it is logical to maintain the 128 byte packet and ½ Full DMA mode for all transactions, on or offline.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

3.3 Protocol - Multi-drop Connection

A multi-drop connection supports more than one sensor in daisy chain fashion on a single channel.



3.3.1 Online

Hub -> Sensors

The Hub inserts timebase sync pulses in the data stream by toggling TxInt for a defined interval. These are received as RxINT at the sensors.

In online mode only, Cypress transmit operation is paced by cell markers and the TXSTOP flag.

The Hub transmits commands to each sensor using the multicast addressing mode provided by the Cypress transceiver. There are a limited number of valid commands in on-line mode.

SEND_DATA
GO_OFFLINE

In online mode, the Hub must continually poll all sensors for data. This is done by sending a SEND_DATA command to the first sensor which readdresses this to the next if it has no data to send, or it sends available data to the Hub at address 00. This sensor must then pass on the next n SEND_DATA commands to ensure that all sensors on the link have an opportunity to transmit their data. This is done as follows:

The Sensor Interface periodically transmits a SEND_DATA command addressed to the first sensor. The period between transmissions is set to a minimum value, but no transmission takes place if the channel

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

receive FIFO ½ full flag is set. The minimum value is set to the worst case latency between the time a sensor receives the SEND_DATA command and the ½ full flag being set in the Hub receive FIFO.

The first sensor either sends available data or increments the address by one and passes the command on.

If the last sensor in the link receives the command and it has no available data, then it increments the address by one and passes this to the Hub. The Hub will discard this command as the address is not matched.

Note: available data means at least 128 bytes in the sensor transmit FIFO.

Sensors -> Hub

The sensors receive commands with the first byte tagged with a Start of Cell Marker. This will be used to initiate the service routine to send data. The sensor data from the previous frame will be sent and then the command packet examined to determine the action requested by the Hub for the next frame.

The sensors never Transmit unless requested by a command from the Hub.

Replies are always 128 bytes in length, addressed to the Hub, with the second byte the sensor serial address.

3.3.2 Off Line

In offline mode the Cypress transmitters are configured in the same manner as in direct connection, except now the sensor address is required.

The Hub writes command strings to the Transmit FIFO with the last byte tagged with a Cell marker. The string is received by the Sensor and the Cell marker asserts the RXSOC – Receive Start of Cell marker. Note that the Hub transmitter is in continuous transmit mode – TXSTOP is deasserted – and the Cell marker is transmitted and is really an end of cell mark in this instance. The Sensor FPGA logic monitors RXSOC and uses this to determine when a complete command string has been received. The command string is read one byte at a time by either the FPGA or microcontroller.

Command strings may extend to 256 bytes.

3.3.3 Loop operation

The Cypress transceiver provides two modes of address matching for loop architectures – multicast where a match of any bit in the address causes an Address Match condition, and unicast wherein all 8 bits must match exactly. In each mode a special Serial Address Marker is transmitted from the Hub followed by a data character which contains the address of the sensor that the following data is intended for. Each Cypress transceiver in the loop compares the address with the contents of its own Serial Address Register. If there is an Address Match, the transceiver writes all subsequent data up to the next Serial Address Marker to its receive FIFO.

The Hydra Hub uses only the multicast format to facilitate both individual addressing and simultaneous transmission of commands to all sensors. This provides a plug and play architecture in which sensors can be replaced or reconfigured at any time and the Hub will recognize the new setup upon a power up sequence.

The 8-bit multicast address is divided as follows:

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

Bit Number	7	6	5	4	3	2	1	0
Sensor	All	6	5	4	3	2	1	0

Sensors are assigned addresses during enumeration, starting at 0, incrementing by 1 in the order in which they are physically connected on the loop. There are 7 possible sensor addresses, with the 8th used as a broadcast address to all sensors on the loop. Sensor number 0 is also used by the Hub itself – If the first command sent to the first sensor reappears at the Hub, then it is apparent that there are no sensors in the link. This also facilitates polling of the link sensors for data when on-line in multi-drop configuration – see later section.

Sensor addresses are added to the Channel number to create the full Logical Sensor Address at the Hub Command level as:

Channel	Sensor	Logical_ Address (decimal)	Transmit_ Serial_ Addr (hex)	Serial_ Address _Register (hex)	Serial Address Register (binary)
0	0	0	0x01	0x81	10000001
	1	1	0x02	0x82	10000010
	2	2	0x04	0x84	10000100
	3	3	0x08	0x88	10001000
	4	4	0x10	0x90	10010000
	5	5	0x20	0xA0	10100000
	6	6	0x40	0xC0	11000000
	All	7	0x80	N/a	N/a
1	0	10	0x01	0x81	10000001
	1	11	0x02	0x82	10000010
	2	12	0x04	0x84	10000100
	3	13	0x08	0x88	10001000
	4	14	0x10	0x90	10010000
	5	15	0x20	0xA0	10100000
	6	16	0x40	0xC0	11000000
	All	17	0x80	N/a	N/a
2	0	20	0x01	0x81	10000001
	1	21	0x02	0x82	10000010
	etc	:	:	:	:

The **Logical_Address** is used to identify each sensor at the Hub Command level within the VxWorks application. It is converted to the Transmit_Serial_Addr in the Hotlink interface driver routines.

The **Transmit_Serial_Addr** is the data character marked with C10.0 and interpreted as an address by the Hotlink receiver on the sensor. It is bit wise ANDed with the Serial_Address_Register value in the Hotlink Receiver to generate an address match.

The **Serial_Address_Register** value is written by the sensor microcontroller during enumeration after power up.

3.3.4 Enumeration

On power up, all the sensors and the Hub default their Serial Address Registers to all 1's, set multicast Address Match mode and the sensors set LOOPTX to true.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

Sensors then reframe when the Hub enables its Hotlink transmitter.

The Hub receiver then reframes completing the physical loop

The Hub broadcasts a CLR_LINK command which resets all the state machines in the sensors associated with the Hotlink operation and receives this itself as a check of the loop integrity

The Hub broadcasts an ENUMERATE_START command, received by all sensors.

All sensors set LOOPTX to false to disable automatic pass on of data

The hub sends the REPORT_SERIAL_NUM command addressed to sensor 0. This command is a table to be filled in by successive sensors filling the next available address with their electronic ID and other information and then passing the table on to the next sensor. The table is initiated with all zero entries.

The first physical sensor in the loop reads the command into its receive FIFO. Sensor logic then 'assigns' number 0 to itself in multicast mode, writes the corresponding bit pattern (0x81) to its serial address register, writes its electronic ID etc to the command table in slot 1 and passes it on with an addressee of 0.

Sensor 1 then asserts LOOPTX to speed future commands and posts an internal ENUMERATED flag

The REPORT_SERIAL_NUM is read by the next sensor which recognises that slot 1 is taken and assigns itself to address 2, writes its electronic ID to slot 2 and passes the command on, etc etc

The end of the operation is signalled by the Hub receiving the REPORT_SERIAL_NUM command, with a table containing the electronic ID's etc of all the sensors and their respective physical locations in the chain.

Once enumeration is complete, all commands from the Hub are sent to a specific address.

All sensors operate in multicast address match mode and LOOPTX wherein data not intended for them is passed through to the next sensor

All replies from a sensor are sent to address 0x00, the Hub receiver, which is set to 'promiscuous' mode to accept all data. All replies are 128 bytes in length to trigger the 1/2FULL FIFO flag at the Hub.

The electronic ID reported in the SERIAL_NUM_REPLY is a non volatile electronic serial number assigned to each head at manufacture. It is intended to index any specific calibration data etc that is required for operation of the system.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

3.4 Hub –> Sensor Command Set

This section describes the commands and data formats used for communication between the Hub and sensors. All commands are assembled in the Hub in response to requests from the Host or internal Hub logic.

The command set and format is the same for direct or loop connections.

The formatting of each command is handled by the Sensor Interface Task in response to commands received from either the Host, Frame Processor, or Hub Control modules.

Byte	Mnemonic	Value	Description
0	Transmit_Serial_Addr	byte	Tagged with (C10.0) Serial Address Marker
1 –> (n-1)	Command and Data bytes	byte	
n	8 bit Checksum of all preceeding bytes	byte	Tagged with (C8.0) Start of Cell

All commands must fit into a single 256 byte packet including header and start of cell delimiting character at the end. Each command may have a different length and format. The Sensor Interface hardware and firmware handles 'packets' only with no knowledge of their contents except the channel to which it should be directed.

The Hub commands supported by the HRPV sensor are as follows –

TABLE 3.4 Hub – HRPV Sensor Commands

COMMAND	Hex	DATA	RESPONSE	Comments
S_RESET	0x01	-	S_STAT_CFG	Forces warm reset of sensor
S_CLR_LINK	0x02	-	S_STAT_CFG	Resets Hotlink Rx state machines
S_REPORT_STAT_CFG	0x03	-	S_STAT_CFG	
S_SET_CFG	0x04	S_STAT_CFG	S_STAT_CFG	
S_SEND_EDGES	0x05	-	-	Broadcast cmd
S_STOP_EDGES	0x06	-	-	Broadcast cmd
S_SEND_RLE_PACKET	0x07	-	S_EDGES	
S_RESET_SYNC_CNT	0x08	-	-	Broadcast cmd – Resets sync count in each sensor
S_ENUM_START	0x09	-	-	
S_REPORT_SER_NUM	0x0A	SER_NUM_TABLE	SER_NUM_TABLE	
S_SEND_VIDEO	0x0B	DET_NUM	S_VIDEO	Data sent from detector DET_NUM directly by Hotlink FPGA logic

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Data structure definitions are:

S_STAT_CFG is a complete description of the 9 detectors comprising each sensor, their status and the status of the pixel processor. This structure is read by the Hub to determine the physical configuration of each sensor and then written back to set threshold and filter values for each detector.

```
typedef struct
{
        short   serial_number;      //      factory assigned unique number in EEPROM
        byte    detector_type;      //      3DM assigned code
        byte    threshold_0;        //      0-255 threshold value for edge detect – detector 0
        byte    threshold_1;        //      0-255 threshold value for edge detect – detector 1
        byte    threshold_2;        //      0-255 threshold value for edge detect – detector 2
        byte    filter_mode_0;      //      spurious pixel filter mode – detector 0
        byte    filter_mode_1;      //      spurious pixel filter mode – detector 1
        byte    filter_mode_2;      //      spurious pixel filter mode – detector 2
        byte    det_status;         //      status
        byte    det_led_vals;       //      led values
        byte    det_cfg_sp;         //      spare
} s_detector_cfg;

struct
{
        byte            sensor_stat;
        byte            sensor_mode;    //      1 == edges; 2 == video
        byte            sensor_temp;
        byte            stat_cfg_sp;
        s_detector_cfg  det_cfg[3];
}
S_STAT_CFG;
```

The S_REPORT_SER_NUM command has a variable length table in the data field defined by the Cell marker. Each sensor reports an 8 byte electronic ID as defined as follows:

Byte location	Description
0	Logical address
1	Factory serial number MSB
2	Factory serial number LSB
3	Firmware part number MSB
4	Firmware part number LSB
5	Sensor Type number MSB
6	Sensor Type number LSB
7	Checksum of preceeding 7 bytes

Sensor type number is assigned by 3DM

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

3.5 Sensor I/F Operation

This section describes the detail operation of the Sensor Interface hardware located on the daughter board. These functions are implemented in the Altera FPGA that provides the interface between the four Hotlink 924DX transceivers and the MPC8260 bus.

Note that channel, Hotlink, and transceiver are used interchangeably for descriptive purposes. Similarly, MPC, MPC8260 and Hub processor.

3.51 Sync

Each channel can be configured to produce a sync at a different divisor of the basetimer frequency. The basetimer frequency is supplied by the MPC. In the initial implementation each channel has an 8 bit value. A value of zero (default power up state) will disable the timer and hence the channel sync pulse. A non zero value enables the channel and is used as a divisor to create individual channel syncs. The MPC selects the phase by writing a value at the appropriate time to each timer after which the timers are gated by the next positive basetimer pulse.

A valid sync pulse is 10 clock cycles based upon the cypress hotlink clock.

3.5.2 Channel Configuration

Normal operation for each cypress transceiver is asynchronous FIFO operation. The FIFOs are always enabled from power on. This allows an external clock to drive both Txclock and Rxclock while the serial link runs off a 16MHz refclk.

Default operation for the cypress hotlink transceivers is asynchronous operation with FIFOs enabled. The transceivers are set to 8bit encoded mode and use a Utopia style interface with flags active low.

To minimize pin count, the transmit data lines from all four channels are wired together. Similarly the receive lines from all four channels are wired together. This means that simultaneous broadcast on all four channels will require loading each channel FIFO in turn, before issuing a global transmit. Flags for each channel are available through the status registers. However, these flags only indicate a most recent update unless the channel is currently accessed.

Default operation for all transceivers is to power up enabled. This gives an immediate indication when connecting cables that a valid connection has been made.

3.5.3 Power up sequence

The FPGA is a dynamic device - its program image is loaded automatically from an Altera Flash on power up (future implementations may be loaded from the MPC). The FPGA is then in a quiescent state.

On power up all four 924DX devices reset autonomously and default to promiscuous mode.

It is assumed that both of the above are fully completed prior to any action by the MPC.

The MPC then performs the following sequence –

Asserts HUB_IO_RESET for a period of 10 microseconds (arbitrary)

Pends for up to 1 second on assertion of IRQ2 by the FPGA to acknowledge a soft reset of the FPGA. This soft reset clears internal FPGA registers and resets all internal FPGA state machines.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

Reads Status_Register_0 to clear IRQ2 – Contents of register are irrelevant

The MPC then proceeds to initialise each channel by issuing the following commands -

Read of Chan_Status_Register[x] to clear any latched values, specifically LFI

Read of Chan_Status_Register[x] to check the current value of LFI

(If LFI is not asserted, this indicates that there is most likely a valid sensor attached to the channel and the following sequence is performed once, else the sequence is performed twice)

Disable Transmitter[x]	*causes assertion of LFI at sensor == reset*
Wait 10 microseconds	*must be at least 24 clocks for 8951 controller at sensor*
Enable Transmitter[x]	*deasserts sensor LFI to initiate warm boot*

Pends for up to 2 seconds on deassertion of channel LFI at Hub. This indicates a successful sensor warm boot and reframe – see below

Reframe request to the channel.

Reset FIFO request to make sure FIFOs are clear.

At this point the channel is considered functional in offline operation.

The Hub LFI* signal is available through a register location – Chan_Status_Register[x]. If the signal has toggled at any time since last read it will set the LFI* bit. The LFI* bit is automatically cleared when Chan_Status_Register[x] is read. This bit set is necessary to ensure that a status check will recognize that a remote transmitter may have momentarily dropped offline which will require at least a new reframe.

Disable Transmitter[x] is accomplished by pulling the external control enable to the ground. This is done through an external decoder (to save pins).

The remote sensor comprises an FPGA device, microcontroller, and Cypress 924DX transceiver. The LFI* signal from this device is routed to the microcontroller RESET pin via a gate enabled by the microcontroller itself. On power up, the microcontroller performs necessary cold boot operations, ensures the FPGA is ready, enables its transmitter, and then enables the LFI* signal routing to its RESET pin.

When the MPC issues the Disable Transmitter[x] command above, the sensor LFI* is asserted, which places the microcontroller in reset from which it emerges when the Enable Transmitter[x] command is set by the MPC. The **warm** boot condition is sensed by the microcontroller from its PCON register and firmware then proceeds to reframe the receiver, and then re-enable the transmitter to signal a successful warm boot.

3.5.4 Reframe

A reframe sequence will first check the LFI* signal to make sure it is currently de-asserted. Discard policy 1 is used for the reframe sequence. The reframe state machine will temporarily bypass the current discard policy and implement discard policy 1. Once the reframe cycle is complete, the discard policy will revert to the original setting. It will then issue a reset of the Rx FIFO. This FIFO reset is slightly different than a regular FIFO reset initiated by the host although it uses the same reset logic. The difference is that once the device is in reset (after 8 clock cycles), it needs to be held in reset until just after reframe is asserted.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

Reframe is held asserted until the Rxempty pin deasserts on reception of the first K28.5 character (will we have to check the data stream for the actual character value). If the reframe is successful the reframe bit that was set by the host will be cleared and the host can read this using a status request. If the reframe fails, the bit will NOT be cleared and the host must clear the bit by writing a FPGA logic reset that will clear the bits associated with commands for that channel and reset all state machines pointed to by that channel. (Alternatively, this could be accomplished by writing a zero to the command request bit – implemented later).

3.5.5 Commands

The following commands are available to MPC8260 via the FPGA. Some commands require that a channel be selected by first toggling txen/rxen while other commands require that they remain deasserted. The command register is currently implemented such that this is set by the MPC using the notr and tx/rx bits of the command register, although this could be changed later.

Reset Receiver FIFO
Reset Transmitter FIFO
Read Status (various locations)
Toggle Transmitter (for remote reset and reframe)
Reframe
Load – the load data command can be used to tell the FPGA to load internal transmit FIFO data into a particular channel. Its primary function is actually to provide an indicator that channel data to the MPC via status read
SAR (load serial address register)
Write to a register location sets the three bits Txsoc,Txsvs,Txsc/d and places them into the internal transmit FIFO(if no write occurs then these bits default to zero for the current data byte).
Transmit data

3.5.6 Endian and Address Issues

The Motorola MPC8260 is a big endian device. The Sensor Interface and sensors are generally little endian. The conversion between the two is done at the physical bus interface between the FPGA and MPC8260 bus.

Internal to the FPGA little endian is used.

All FPGA registers available to the MPC are accessed as 32 bit even though they may occupy less physical space, to avoid the confusion of byte selects.

All address decodes are to 16 byte boundaries to facilitate data transfer. Note that this is assumed as little endian also. Translation to big endian occurs on the schematic when connecting the FPGA outputs to the MPC bus. References in this document to the FPGA will show channel decode internally as little endian with the equivalent little endian address lines. E.g a[7..4] could be used to decode to 16 byte boundaries in little endian format while in big endian this becomes MPC A[24..27]. Note that all data groups are written MSB..LSB.

In big endian reads of memory locations in the MPC8260, the following table shows the bit, 8 bit CHAR, 16 bit SHORT and 32 bit WORD mapping

Size	Address (Base 0x2300 0000)																													
bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHAR	0x2300 0000								0x2300 0001								0x2300 0002								0x2300 0003					
SHORT	0x2300 0000																0x2300 0002													
WORD	0x2300 0000																													

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited excepl as 3DM Devices Inc. may otherwise agree to in writing.

3.5.7 Device Selection and Register Decode (little endian)

In the single FPGA device implementation CS8# will be used to select the FPGA. A secondary FPGA device could reside at CS8# plus a9 =1 or CS9#.

EQ	R/W#	a8	a7	a6	a5	Registers
0	0	0	0	0	0	
1	0	0	0	0	1	Control Write
2	0	0	0	1	0	Command Write
3	0	0	0	1	1	TxData Write d[7..0]
4	0	0	1	0	0	TxData Write d[10..8]
5	0	0	1	0	1	Sync Channel 0
6	0	0	1	1	0	Sync Channel 1
7	0	0	1	1	1	Sync Channel 2
8	0	1	0	0	0	Sync Channel 3
9	0	1	0	0	1	Serial Address d[8..0]
10	0	1	0	1	0	
11	0	1	0	1	1	
12	0	1	1	0	0	
13	0	1	1	0	1	
14	0	1	1	1	0	RESERVED
15	0	1	1	1	1	Reset – fpga state & registers
16	1	0	0	0	0	
17	1	0	0	0	1	Control Read
18	1	0	0	1	0	Command Read
19	1	0	0	1	1	
20	1	0	1	0	0	
21	1	0	1	0	1	
22	1	0	1	1	0	
23	1	0	1	1	1	
24	1	1	0	0	0	Status FPGA
25	1	1	0	0	1	Status Channel 0
26	1	1	0	1	0	Status Channel 1
27	1	1	0	1	1	Status Channel 2
28	1	1	1	0	0	Status Channel 3
29	1	1	1	0	1	
30	1	1	1	1	0	Read FPGA Receive FIFO
31	1	1	1	1	1	

Note: The Vision Probe automatically does read verifies after writes. This can cause a problem for fifos therefore some locations are RESERVED.

3.5.8 Channel Data Transfer to Memory

There are two mechanisms for transferring data to the MPC8260 from the Hub I/O board:

DMA – utilizing bursts of 8 consecutive 32bit data.
Interrupt (or direct) – utilizes single 32bit reads.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

All transfers are initiated by assertion of a HALF# full flag by a channel. This is recognised by the FPGA logic in its round robin polling and starts the following process:

FPGA logic write 4 bytes of header to the FPGA output buffer. These 4 bytes provide the source and destination routing for the following data and general status information. The actual format is described later below.

128 bytes are then transferred from the channel Hotlink receive FIFO to the FPGA output buffer.

DMA Transfers

The FPGA asserts a DMA request on DREQx.
The MPC8260 reads the 4bytes of header information using a single 32 bit read (non-DMA).

The MPC8260 then asserts DACKx which results in immediate transfer of 132 bytes from the FPGA output buffer utilizing burst transfers.

The FPGA signals completion by asserting DONEx and pends on DACKx assertion by the MPC to complete the handshaking.

The FPGA then resumes round robin polling.

The DMA transfer mode – burst, block size etc. - is setup in the DMA BD control registers and is subject to change during development.

Internally, the MPC maintains two chained BD buffers, 0 and 1, which transfer data to their respective memory buffers, BUF0 and BUF1. The VxWorks code continually monitors for writes to these locations by the DMA routines. Completion of a DMA transfer to either buffer releases a higher level routine to read the buffer and move the data to the correct unpacked data buffer as determined from the first 4 header bytes. The two buffers, BUF0 and BUF1 provide sufficient latency to support continuous reads from the FPGA buffer, noting that data is transferred from the Hotlink Rx FIFO to the FPGA buffer at 33 Mbytes/sec, but the DMA transfer takes place at 4x this using 32 bit wide data, and the secondary BUF0/1 write and read operations occur at a 66 MHz bus speed on 32 bit wide data.

Interrupt Transfers

The FPGA asserts an interrupt request. The MPC8260 then uses single 32bit reads to transfer 132bytes of data. The interrupt line from the FPGA will automatically de-assert when the FPGA internal buffer is empty.

3.5.9 MPC to/from FPGA Bus Transfers

TS# = transfer start (asserted by MPC) with appropriate address lines
BDIP# = burst device in progress – deasserts on second to last data transfer
SDACK# = asserted by MPC to indicate DMA transfer from FPGA.
Intmpc = interrupt to MPC from FPGA to indicate 128bytes available for transfer.
R/wr# = MPC read write control.
Data(16..31)
Address[8..5]

Note that because all registers decode to 32bit locations, byte controls and byte level addressing is ignored by the FPGA.

For transfers of channel data to/from the channel FIFOs this will be a burst capable transfer. All other reads and writes to the FPGA will be forced to single transfers. All bus transfers begin by decode of CS8 and the

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

appropriate address lines with the assertion of TS# from the MPC. The FPGA will respond by taking control of the data bus as necessary and by the appropriate assertion of WAIT.

The only exception to this transfer selection is an SDACK acknowledge to a DMA request. The address on the bus at this time is usually the memory address access and therefore the FPGA will respond to SDACK & CS8 only for this transfer.

The FPGA assumes tristate control of the appropriate interface pins once it detects the appropriate access true.

READ (single access from internal 16bit register):
For a read transfer, the MPC must:

-set r/wr# to read, active high
-assert /CS8 and a valid address (these must be valid during the /TS assertion)
-with a valid address and /CS8 asserted, assert /TS for one clock cycle to signal valid address available
-/bdip must remain de-asserted as single access reads are not burstable
-release control of the data bus
-release control of the /TA line

The FPGA will respond:

-the FPGA will latch and decode the address when it sees /TS asserted
-the FPGA will take control of the /TA line once it decodes an access to itself
-the FPGA will also take control of the data bus
-once it has driven the data bus valid, it will assert /TA for one clock cycle to indicate that the requested data is ready.
-the data bus and /TA will then be tristated.

WRITE (single access from internal 16bit register):
For a read transfer, the MPC must:

-set r/wr# to write, active low
-assert /cs8 and a valid address
-place data on the data bus (this can occur in the next cycle after /TS assertion)
-with a valid address and /cs8 aserted, assert /TS for one clock cycle to signal valid address available
-wait for /TA response from FPGA before releasing data bus.

The FPGA response:

-the FPGA will latch and decode the address when it sees /TS asserted
-the FPGA will take control of the /TA line once it decodes an access
-once the data has been latched in the appropriate register, the FPGA will assert /TA.
-/TA tristates after de-assertion.

DMA Transfer (BURST READ – 32bit access)

The lead off transfer is 5 clock cycles followed by two cycles for each subsequent transfer.

3.5.10 Bus decoupling

The Cypress Hotlink Channel bus will be driven by a clock that is asynchronous to the MPC bus clock (clkout). An asynchronous implementation was chosen to improve the physical layout and to provide flexibility for different MPC processor speeds. The decoupling or recoupling (depending on your point of view) of the two busses is accomplished internally to the FPGA.
Decoupling is necessary for data transfers to/from the channels and for writes to internal registers to avoid metastability. Reads from internal registers do not require decoupling as there is no metastability issue.

For data transfer between the two busses, internal FPGA dual clock FIFOs are used. There is one FIFO for receive data and one for transmit data. Reads and writes of the internal FPGA FIFOs will be sequential to ensure that the MPC knows when data is available or has been sent. E.g. after the interrupt to the MPC 128

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

bytes of data will be written to the internal Receive FIFO. No other writes to the FIFO will occur until the MPC has read 128bytes from the FIFO. The MPC can verify that the FIFO is full by checking the bit in the status0 register. Reset commands of the internal FIFOs are available although this could be setup to occur automatically. Similarly for writes to the transmit channels. The MPC will place data in the internal FIFO. Once it has transferred all the data to the internal transmit FIFO, it will then issue a load command to load the data into the channel FIFO. The MPC monitors the command register bit until the load command bit is cleared, at which time it can directly issue transmit commands to the channel FIFO as necessary.

Decoupling of the internal registers is also necessary to avoid metastable conditions. This will be initially handled by a three cycle delay of TA# externally (later moved internally).

3.5.11 State Machines

There are two main state machines inside the FPGA. One handles the interface to the MPC. The other controls the polling of each hotlink channel and command execution. See the attached flow charts for the details of these and all lower level state machines.

MPC State Machine: Handles all interface controls read/write betweent the MPC processor bus and the FPGA internal registers/channels.

Cypress Polling State Machine: This state machine spends most of its time examining the bits in the control register.

Pending bit – indicates that a valid command awaits execution in the command register. The pending bit is cleared by the state machine when the command has finished execution.

Repeat bit – indicates that the current command should be continously repeated until this bit is cleared by the MPC8260.

Daisychain pause – For a star configuration, this bit should be set to zero. Set to 1 indicates that the current token in the transmit buffer is invalid. The polling state machine will halt execution until this bit is cleared. Setting of this bit is only necessary if there is an online daisychained channel in the system and the MPC8260 wishes to temporarily interrupt operation to manually send commands.

b[12..9] – these are the channel select bits. Only one bit will be set a time. Pending commands will be executed on the channel that is selected.

b[4..1] – these are the channel active bits. A set bit indicates that this channel has been properly initialized and is available to be polled.

Most of the time, the state machine polls the pending bit and the active channel bits (b4..b1). If a channel bit is set in the control register, this indicates that this is an active channel. The state machine will then select the external hotlink channel and poll the channel flags. If a bit is not set in the control register, then this channel is ignored. The bits are set by the MPC following a successful poweron, reframe sequence, etc of the channel. Note that the polling state machine will poll one channel bit of b4..b1 followed by a poll of the pending bit and subsequently a poll of the next bit in sequence b4..b1.

After poweron the control register defaults to all zeros which causes the polling state machine to cycle endlessly checking for a valid bit but not executing anything since all bits are set deasserted.

Commands can be entered by the MPC by first entering the command into the command register. Secondly, the MPC sets the pending bit in the control register while setting a corresponding channel bit in bits b[12..9]. The polling state machine will recognize the pending bit, select the appropriate channel based on bits b12..b9 and

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

then execute the command. Command execution results in the MSB of the command register cleared and then finally the pending bit is cleared by the polling state machine which then returns to polling.

The Repeat bit can be used by the MPC to cause a repeat of the current command.The sequence for repeating a command is: First write the command to the command register, then set the pending bit and repeat bit and the channel to be selected (b12..b9) into the control register. The polling state machine will detect the pending bit and execute the command and clear the MSB of the command register as well as the pending bit. The polling state machine will the check the repeat bit and re-execute the command until such time as the repeat bit is cleared.

3.5.12 CONTROL REGISTER (little endian)

BITS	definition	MPC access	FPGA access
15 (msb)	1=pending	Set/R	R/Clr
14	1=repeat	W/R	Read only
13	1=daisychain pause		
12	1=Channel 3 select	W/R	Read only
11	1=Channel 2 select	W/R	Read only
10	1=Channel 1 select	W/R	Read only
9	1=Channel 0 select	W/R	Read only
8			
7			
6			
5			
4	1=Channel 3 active	W/R	Read only
3	1=Channel 2 active	W/R	Read only
2	1=Channel 1 active	W/R	Read only
1	1=Channel 0 active	W/R	Read only
0 (lsb)			

Bits b4..b1 are set by the MPC to indicate to the FPGA which channels are active and therefore should be polled. Bits b12..b9 are used by the MPC to select a channel and execute a command on that channel.

Pending bit indicates that there is a command waiting to be executed in the command register. Pending bit is cleared by the FPGA when the command is finished.

Repeat bit indicates to repeat the previous command. (currently not fully implemented)

Daisychain pause bit is used to temporarily interrupt an online daisychained channel.

3.5.13 COMMAND REGISTER (effectively two registers here controlled by expand bit)

Due to the muxing of the four channels and to simplify the initial implementation of the FPGA logic, the control of /txen and /rxen was moved to the command register. Therefore if a command requires the assertion of these control signals, that information must be sent with the command. A higher level state machine will assert these signals as required. The lower level command implementations then adjust to the timing necessary for a partcular command. In a later implemenation, the commands could be referenced to a lookup table for /txen,/rxen requirements.

All commands must be sent with the command done bit asserted. When the command has completed, the logic will clear the command done bit. The 8260 can poll the command register status at any time to determine the command that is in progress. When a command successfully completes, it will clear the pending bit in the

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited excepl as 3DM Devices Inc may otherwise agree to in writing.

control register. If a command times out, the 8260 can reset the command logic by writing to the "RESET – fpga state & registers" location as indicated in the address decode table. This will reset the polling subroutine and the clear the control and command registers to zero.

Primary Command Register

BITS	definition	MPC access	FPGA Access
15	0=Command done	W/R	R/CLR
14	1=No txen/rxen	W/R	Read only
13	0= txen,1=rxen	W/R	Read only
12	Expand = 0	W/R	Read only
11	1=tx fifo reset	W/R	
10	1=rx fifo reset	W/R	
9	1 = reframe request	W/R	
8	Rxmode[1](default =1)	W/R	
7	Rxmode[0] (default =1)	W/R	
6	1=policy change	W/R	
5	1= set address register	W/R	
4	1=load data to tx FIFO	W/R	
3	FOTO(1=on,0=off)	W/R	
2	0=/TxBist enabled	W/R	
1	0=/Rxbist enabled	W/R	
0	1=haltcontrol	W/R	

Alternate with expand bit = 1

BITS	definition	MPC access	FPGA Access
12	Expand = 1	W/R	Read only
11	1=reset internal fifos	W/R	
10	Latch LED	W/R	
9	dlb[1](0=default)	W/R	
8	dlb[0](0=default)	W/R	
7		W/R	
6		W/R	
5		W/R	
4		W/R	
3	1=Transmit all channels	W/R	
2		W/R	
1		W/R	
0		W/R	

COMMAND	VALUE written to command register	COMMENTS
/TXRST	C80E	
/RXRST	C58E	
REFRAME	C38E	
POLICYCH (latch values)		
/TXBIST & /RXBIST	A188	exit by RESET
LOAD (HL from fpga tx fifo)	819E	qualify with type bit
TRANSMIT_ALL		
DAISYCHAIN ONLINE		
POLL FLAGS	C18E	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

/RXEN (transfer HL data to fpga fifo)	A18E	use for debug only

3.5.14 FPGA Status Register

BITS	definition	MPC access	FPGA Access
15	1=FPGA Rcv fifo is full	Read only	
14	1=FPGA RCV fifo is empty	Read only	
13	1=FPGA Tx fifo is full	Read only	
12	1=FPGA tx fifo is empty	Read only	
11		Read only	
10		Read only	
9		Read only	
8	1=Txsync3 interrupt	Read only	
7	1=Txsync2 interrupt	Read only	
6	1=Txsync1 interrupt	Read only	
5	1=Txsync0 interrupt	Read only	
4	1=Interrupt on channel 3	Read only	
3	1=Interrupt on channel 2	Read only	
2	1=Interrupt on channel 1	Read only	
1	1=Interrupt on channel 0	Read only	
0		Read only	

3.5.15 Channel (3..0) Status Registers

BITS	definition	MPC access	FPGA Access
15	0=Link fault	Read only	update
14	1=reframe done, 0 =not done	Read only	update
13	1=RXRVS	Read only	Update
12	0=Rxfull#, 1=not full	Read only	Update
11	0=Rxhalf#, 1 = not half	Read only	update
10	0=Rxempty#,1=not empty	Read only	
9	0=Txfull#, 1 = not full	Read only	
8	0=Txhalf#, 1 = not half	Read only	
7	0= Txempty#,1 = not empty	Read only	
6		Read only	
5	1=channel daisychained & online	Read only	
4		Read only	
3		Read only	
2		Read only	
1		Read only	
0		Read only	

3.5.16 Sync Registers [3..0]

Currently these are 8bit values but could probably be changed to 16 bit. When all zero that means that the external sync is disabled. A non-zero value will enable the sync on the next MPC basetimer pulse.

3.5.17 Transmit Data Buffer

The Hotlink transmit data bus is 11 bits wide. Data is first written to the internal transmit buffer and then loaded into the Hotlink channel FIFO. Note that the internal transmit buffer will be 128x11bits. Data bits d[7..0] are the

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

normal data bits and are written to the TDATA location. Data bits d[10..8] are written to the TTDATA location as shown in the decode table. To minimize overhead the following sequence should be followed:

When writing normal data, simply write to location TDATA as often as necessary. The upper bits will default to zero after a poweron and will also default to zero after a TDATA write. Therefore an 8bit write to location TDATA will cause an 11bit write to the internal transmit FIFO with the upper bits set to zero. To write special characters, first write to the TTDATA location and then write to the TDATA location. This will cause a full 11bit character write to the internal FIFO with the upper 3 bits set as desired. This is a legacy implementation; a single 16bit write by the MPC8260 could be used in a later implementation.

The internal transmit buffer (dual port RAM) decouples the two interfaces which has benefits both for differing clocks and differing write timings. This would allow the hub to load data to appropriate channel at an appropriate time in online operation. Future implementations could allow simultaneous receive of data while load the internal FIFO but this is not supported at this time. Currently the main benefit is that the internal FIFO can be written to at any time by the MPC8260 without stopping the link functions. The internal FIFO allows variable length packet sizes.

3.5.18 Serial Address Register Access

The Serial address register is accessed through the RXDATA bus. RXRST# must be asserted to access the Hotlink internal register. Serial address register access complicates matters for RXEN* because it must be asserted as a data strobe into the Hotlink channel rather than its usual function as data read enable. Also the assertion of RXEN* must follow the assertion of RXRST#. RXRVS becomes a R/W# input for this access and RXSC/D# =0 selects multicast; = 1 selects unicast.

For our implementation, an internal 9bit register d[8..0] location is decoded internally. The ninth bit represents the RXSC/D bit. The sequence for a Serial address change is to write to the internal address location as indicated in the decode table. Then write the serial register access command to the command register. Finally set the pending bit and the channel selected into the control register and the internal state machine will proceed with the Hotlink access.

3.5.19 TXSTOP# ,TXHALT# and data transmission

Given that the last byte is tagged by TXSOC, Txstop# can not be used for data transmission control. Therefore, TXSTOP# is pulled up for all 4 channels. In order to support simultaneous transmission of a command on all four channels, TXHALT# is used instead.

For data transmission of commands from the hub, the command LOAD is used in conjunction with the bit haltcontrol. For simultaneous transmission of a command on all four channels the command TRANSMIT_ALL is used after the data has been loaded into each channel transmit FIFO.

The default on powerup will be "star configuration" mode for the hub. Once a channel has been initialized, it is available for manual LOAD commands. The bit, haltcontrol, is defined in the command register and controls the type of manual LOAD command in progress. If the "manual LOAD" is in preparation for a "transmit all" command, then the bit must be first set to 1. If the bit is set to 1 when a manual load command is issued, /txhalt will be set to GND (transmission halted). Once the manual LOAD has completed (i.e. the command has been loaded into hotlink channel FIFO), the /txhalt control state machine will then await a TRANSMIT_ALL command. Once a txall command is received, /txhalt = VCC will be set and the command will be sent. The default setting is haltcontrol=0 which means /txhalt=VCC and data loaded into the hotlink channel FIFO flows through continously. This mechanism is required because active channels must be accessed individually in order to load commands into the cypress hotlink transmit FIFOs before issuing a simultaneous command to all channels.

3.5.20 Unused and unsupported pins

Unused = these pins have a defined and reserved location on the Altera FPGA but have no associated logic internally.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

Unsupported = these pins have NO defined location on the Altera FPGA. They may have external logic,etc.

LOOPTX,TXSTOP#,RXDATA[9] are unsupported on the hub Hotlink interface.

3.5.21 Fanned out pins

These pins are fanned out in a second smaller device to save pin count on the main Altera device.

The following pins are fanned out externally:

/RXHALF	fans out to	/RX4HALF, /RX3HALF, /RX2HALF, /RX1HALF.
/TXHALF	fans out to	/TX4HALF, /TX3HALF, /TX2HALF, /TX1HALF.
FOTO	fans out to	FOTO4,FOTO3,FOTO2,FOTO1.

3.5.22 Interrupts

There is one interrupt line from the FPGA to the MPC. It asserts when there is a packet of data in the FPGA internal Receive FIFO.

3.5.23 Encoder

Currently the encoder count is pass through only and is not used by any of the existing logic.

3.5.24 Output Data Bus

Altera FPGA architecture does not support tristate internal bus where one common output is fed by more than one tristateable input. The result is that muxes had to be used to provide register reads. This increases resource utilization as well as propagation driven pipelining. To minimize the timing issues with propagation driven pipelining, three stages of muxes were used. The deepest stage muxes that status register outputs. The second stage muxes the status register mux output with the control and command registers. The final stage muxes the single register output with the rcvfifo output. Need to recognize that changes may need to be made to this architecture to handle aborted DMA transfers from the internal rcvfifo because due to pipelining there may still be valid data in the pipe.

3.5.25 BIST

To save pins, all /TXBIST pins are wired together as are all /RXBIST pins. When BIST is asserted all channels will be in BIST mode simultaneously. However, in order to monitor the RXRVS and other flags, the channel needs to be addressed and selected. I.E. for RXRVS both /AM and /RXEN must be asserted for that channel.

3.5.26 Data Throttle - Sync Pause

A stretched sync pulse is used to stop a channel from sending any more data to the hub. This occurs when the hub hotlink channel fifo flag asserts full.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

The length of a valid sync pulse is 10clock pulses (cypress hotlink fifo clock). The length of a pause condition is minimum 25clock pulses.

A pause condition will not be removed until the cypress hotlink fifo is completely empty. This means that two DMA transfers are required to service a paused channel. On pause de-assertion, the pixel processor will set its transmit fifo to continous and immediately send 256 bytes into the link as it drains its FIFO. The pixel processor will then resume transfer data after the next vsync. During a long pause condition, the internal sync generator in the pixel processor will take over such that when data transfer resumes no sync counts have been lost relative to other channels.

3.5.27 Data Packet

All data received by the hub is packetized into 128bytes. The first 126bytes are valid data. The final two bytes are identical status bytes. This particular implementation is partially due to a bug in the Cypress Hotlink transceiver.

Status byte:

BITS	definition	MPC access	FPGA Access
7			
6			
5			
4			
3	1=default (reserved)		
2			
1	1=paused		
0	1=halted		

a) Paused – if this bit is set it indicates that a sync pause occurred during this last 128byte transfer.
b) Halted – if this bit is set it indicates that an overflow occurred on the pixel processor. An overflow occurs when the pause duration results in the pixel processor filling its hotlink transmit FIFO.

3.5.28 Daisy Chained Configuration

For daisychain configuration, the DAISYCHAIN command is required to start the auto token loading process. The system will power up defaulting to the star configuration. The procedure is to initialize a channel in the default "star type" configuration, then write the token into the FPGA transmit buffer and issue the daisychain command with the appropriate channel selected in the control register. The command will load the first token into the hotlink fifo, send the first token, and then set the daisychain bit for that channel in the channel status register. All subsequent tokens are auto-loaded into the channel when an incoming response has been received into the internal FIFO. Therefore the next token is loaded once the polling routine has found a channel with incoming data, and begins transferring that data into the fpga internal receive FIFO. This requires leaving /AM for that channel selected until both the token is loaded and the receive data transferred. Since the size of the token/incoming data is equivalent, this should provide minimum overhead to the system speed. The service request to the MPC8260 will be issued once the token load/receive data transfer have both completed. The token will not be sent until the transfer of receive data to the MPC8260 commences.

A combination of star and daisychain can be supported. Commands can be issued to any channel as before, however, the token must be rewritten into the fpga transmit buffer by the MPC8260 after the command has completed. This requires implementation of a daisychain_pause bit. This be must be set in the control

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

register when the pending bit is set. After the command is executed, the pending bit will be cleared as before, but the state machine will not resume polling until the daisychain_pause bit has been cleared by the MPC8260. If the MPC8260 wishes to receive a response from a channel before resuming normal polling, it can either:

a) Temporarily clear active channel bits while leaving the channel in question asserted. Then clear the daisychain_pause bit and let the state machine auto-find the channel response (sit polling on a channel). Once the MPC8260 is finished it must re-write the correct token before resetting the active channel bits to their previous values. (recommended)

b) or the MPC8260 can manually check flags and manually issue a read channel command /service the channel before writing the token / clearing the daisychain_pause bit.

If there is more than one daisychained channel (from the hub fpga viewpoint), then there is a limitation that the token must be the same for each daisychained hub channel. The ideal solution would be to have multiple tokens stored in the fpga that could be muxed and loaded on a channel specific basis. However the resource usage may be prohibitive for the current device. A limited version which alleviates resource usage could have channel specific storage locations that are only one byte. Then the fpga would auto-write the channel specific data to a byte location in the DRAM token before auto-loading the token.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.0 Hub Firmware - VxWorks

4.1 Overview

The Hub operating system is implemented using the VxWorks multi-tasking package developed using Tornado II under Win98/NT. VxWorks provides all the real time kernel functions, socket and TCP/IP support and access to all the required peripherals on the MPC8260.

The VxWorks system image is loaded on power up by a boot block loader resident in Flash/ROM. The 'image' may comprise both the VxWorks OS and the application code to support all specific Hub functions. See Appendix I for details.

The following sections describe in point form the grouping and functionality of the VxWorks tasks that comprise the Hub operation. Each section also lists the Host commands that are accessible through the relevant TCP/IP socket.

Coding is done in 'C' and follows the conventions described in *VxWorks Programmer's Guide – Appendix I – Coding Conventions*. The source code is built using the Project facility in Tornado II with the following key parameters:

Project type:	Downloadable VxWorks module
Toolchain:	PPCEC603gnu

The resulting output module is loaded as described in Appendix I.

A single header file – hubHostConstants.h – defines the port values, command constants, and other values required by both the Host and Hub software. This header file is an integral part of the VxWorks system image. The values therein are readable via the hubConsole socket which is located at port number HUB_CONSOLE_PORT. Important values are also displayed during boot on the RS232 connection.

4.2 Hub Control Tasks

The Hub Control tasks provide the following functions:

4.2.1 Hardware boot and self test

The MPC8260 motherboard boots from ROM based VxWorks in the 4MB SIMM flash referenced by CS6. This initialises drivers etc and then attempts an ftp file load using the Host attributes contained in flash (user configured via RS232). The default ftp file name is vxworks which should contain the application program, as generated from the Tornado II environment. The ftp server is the default supplied by WindRiver, *wftpd32.exe*

This is the default VxWorks boot sequence. It is intended that production releases of the hub product will use a complete ROM based boot with the program image contained in on-board flash memory for completely standalone boot up.

See Appendix I for specific VxWorks boot implementations.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

4.2.2 VxWorks boot

The conventional network based boot sequence is fully described in the VxWorks Programmer's Guide, Section 8.3 – VxWorks Initialisation Timeline. The alternate ROM based sequence is described in section 8.6.3

In the conventional case, control is eventually passed to usrRoot() in usrConfig.c which finally calls the **hubStart()** task which spawns all subsequent tasks described below.

The VxWorks module that controls the hub operation is called hctrl

4.2.3 Hub Start Up – Application entry point

The task hctrlStart() is the main entry point to the application code and performs the following –

Prints boot up message to console
Creates hctrlConsole socket
Spawns hctrlConsole task
Creates hctrlControl socket
Spawns hctrlIO task
Spawns sensor and frame processor tasks as determined by enumeration and Host system configuration requests / data

The hctrlConsole task provides back-door access to commonly used functions on the Hub. hctrlStart() creates a socket at the CONSOLE_PORT_NUM and adds it to a list that is polled with **accept** A successful connection spawns the hctrlConsole task and two semaphores, console **********

4.2.4 hubIO Task

This task handles all the hub IO functions. It initialises the hardware, enumerates the attached sensors, and maintains the sensor status and configuration tables.

4.2.5 Hub_IO start up and Channel intialisation

Section 3.5.3 describes the hardware sequence. The hubIOStart() function issues the appropriate commands for this sequence and reports the results in the sensor status table.

Enumeration
Spawn sensor tasks as per configuration table

Spawn frame processor tasks as per configuration table

Sensor Interface Handler
Command assembly
DMA handler

Other I/O handler (bits, encoder, LEDs, serial port(s))
Watchdog

History log of start up – errors – operator actions etc
Summary overview of connected sensors, operating tasks, resource useage etc
Diagnostics

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

4.3 Sensor Tasks

Status Read
Configuration of sensor
Read of calibration coefficients
Configuration changes
Diagnostic data tasks – video, raw data etc

4.4 Frame Processor Tasks

On/Off line switching
Data assembly and conversion

4.5 Console Tasks

The console task is a single VxWorks task that provides a simple interface to the majority of the Hub functions. It is accessed via a single socket and is intended as a telnet client providing simple printf style output to a user screen.

Once connected, in response to a terminal key press, the task prints a list of user options.

4.6 Error Handling

4.7 Diagnostics

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

5.0 Hub Host Commands

5.1 Overview

The Host/Hub connection is implemented using an Ethernet physical layer and TCP/IP protocol. Each of these layers adds routing and checking information as described in a later sub-section.

The **Hub** is the **server** and the **Host** is the **client**.

The Host requests information and the Hub provides this. The Hub never provides information unless requested.

Each communication transaction then consists of the Host issuing a request and the Hub replying, possibly after a delay, with an appropriate response.

The Host request is termed a COMMAND, and the Hub reply a RESPONSE. COMMAND and RESPONSE are forms of message.

The message formats have been designed to facilitate addition of new commands, sensor types etc with minimal changes to the low level driver code responsible for communications at the Hub and Host ends. To this end, a brief overview of the socket communication configuration is in order -

The server and client open a socket communication path and agree upon protocol and maximum and minimum sizes for messages.

The server and client allocate receive buffers set to at least the maximum message size - larger if more than one message of maximum size could be received before the handler deals with it.

The server and client each spawn appropriate handlers for each socket.

The handler, periodically or on an interrupt basis, accesses the receive buffers for data, assembles this into complete messages, and makes this available to higher level routines.

The handler must know the number of bytes in the message in order to signal receipt of the complete message to the higher level tasks. This length is specific to each message type and could be passed to the driver for each message and the expected response. This mechanism is clumsy and makes system upgrades more complicated. The preferred approach is to add a header containing the message length and make this header mandatory for all messages. The low level drivers can thus handle any present and future message with no knowledge of their contents

5.1.1 Message Format

All Hub messages are defined in the header file HHMessages.h

All messages begin with the same 4 byte header assigned as follows:

typedef struct **T_SKT_MSG_HDR**

type	Variable	Units	Bytes	Value
uchar	seqID	-	1	Unique count assigned by Host

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited excepl as 3DM Devices Inc. may otherwise agree to in writing.

uchar	COMMAND/RESPONSE	-	1	
ushort16	msgLength	bytes	2	Total number of bytes **not including** the size of this header in the COMMAND or RESPONSE
			4	

The total number of bytes is limited to the 16 bit value - 4. This is a conscious design decision for the following reasons -

Receive buffers can be allocated as 64k bytes

64k bytes is a reasonable size for a single transaction given that no other messages can be inserted in this block of data, unless out-of-band signalling is used.

Larger messages can be broken up into this segment size.

5.1.2 Command Format

Commands consist of the single byte COMMAND, and then the DATA portion. These are defined in the tables below. There is no checksum or other error checking as there is adequate provision for this in the outer TCP/IP and Ethernet layers.

5.1.3 Response Format

Responses from the Hub consist of the single byte COMMAND echoed back to the Host, and then the requested DATA portion. There is no checksum or other error checking as there is adequate provision for this in the outer TCP/IP and Ethernet layers.

5.1.4 Data Format and Sizes

There are a number of considerations when defining data, structures, and messages:

Big/Little Endian
Default structure packing by host compilers
Minimum data packet for Ethernet communications

All data structures are packed out to 4 byte boundaries. This is driven by the 4 byte message header structure described above, which in turn is driven by the nominal 70 byte target payload - see below - which can usefully comprise 64 bytes + 4 bytes of header. *(confused yet?)*

The sizes of the data structures have been defined with consideration for the data payload capacity using TCP/IP protocol carried on an Ethernet connection.

The Ethernet standard defines packet sizes of 64, 128, 256, .. 1518 bytes. (see 8260UM.pdf Chapter 30 Fast Ethernet Controller) Each packet has a fixed 14 byte header (DESTINATION/SOURCE/TYPE) and 4 byte CRC trailer leaving 46, 110, 238, .. 1500 bytes etc respectively for the TCP/IP information. The TCP and IP headers each comprise a minimum of 20 bytes (no Options), for a total of 40 bytes leaving useable DATA payload of 6, 70, 198, ..1460 bytes etc respectively.

Obviously, if the Ethernet link is the pacing link in the communications path, then a single large message of up to 64 bytes is much more efficient than say 4 messages of 16 bytes - actually at least 4x as fast. Based on this,

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

the status and configuration messages have been defined as what may appear to be relatively large structures, with a typical length of 64 bytes max + 4 bytes for the message header added to this.

5.2 Hub Socket Commands

These commands comprise the command set available via the Hub Control Socket connection. The Hub Control Socket is always available for connection and is the starting point for any configuration changes.

The HH_REPORT_CFG is the only means to ascertain what sensors are connected and what frame processor resources are loaded in the Hub firmware.

A typical configuration session starts with a HH_REPORT_CFG request from the Host, after which the Host can query sensors on each channel and use the HH_LAUNCH_FP command to request opening of a Frame Processor socket followed by messaging with the FP.

COMMAND	Hex	DATA	RESPONSE	Comments
HH_REBOOT	0x01	-	hubStat	
HH_SET_CFG	0x02	hubCfg	hubCfg	Not available in online mode
HH_REPORT_CFG	0x03	-	hubCfg	
HH_REPORT_STATUS	0x04	-	hubStat	
HH_RESET_ENC_CNT	0x05	-	hubStat	
HH_SET_CHNL_CTRL	0x10	chanCtrl	chanStat	Not available in online mode
HH_GET_CHNL_STAT	0x11	chanNum	chanStat	
HH_LAUNCH_FP	0x20	fpNum	hubStat	
HH_KILL_FP	0x21	fpNum	hubStat	
HH_GET_FP_IDENT	0x23	fpNum	fpIdent	
HH_LAUNCH_SENSOR_TYPE	0x30	sensorType	hubStat	
HH_KILL_SENSOR_TYPE	0x31	sensorType	hubStat	Not available in online mode
HH_GET_SENSOR_TYPE_IDENT	0x32	sensorType	sensorTypeIdent	
any	0xFF	-	MSG_ERROR_CODE	Returned in response to an erroneous message

where

T_HH_CFG **hubcfg;**
T_HH_STAT **hubStat;**
T_CHNL_CTRL **chanCtrl;**
T_CHNL_STAT **chanStat;**
uchar **chanNum; /* 0..7*/**
uchar **fpNum;**
T_FP_IDENT **fpIdent;**
T_SENSOR_TYPE_IDENT **sensorTypeIdent**

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

uchar **sensorType**

Data structures are as follows:

typedef struct T_HH_CFG /*Hydra Hub Configuration */				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uchar	sensorPerChannel[8]	n/a	8	Num sensors on each channel
uchar	sensorTypeChannel[8]	T_SENSOR_TYPE	8	Type of sensor on each channel
uchar	numFrameProcessors	n/a	1	Required for get/set FP_CFG
uchar	pad[15]	n/a	15	
			32	

typedef struct T_CHNL_CTRL /* Channel Control requests sent by Host */				
type	**Variable**	**Units**	**Bytes**	**Value**
uchar	chnlNum		1	0..7
uchar	chnlCmd[0]		1	see table
uchar	chnlCmd[1]		1	
uchar	chnlCmd[2]		1	
uchar	chnlData[0]		1	see table
uchar	chnlData[1]		1	
uchar	chnlData[2]		1	
uchar	chnlData[3]		1	
			8	

enum **chnlCmd** /* Channel Commands - Low level access to Cypress Transceiver */

size defaults to int == int32

Constant	Value	Description
clearCommand	0	Clears commands in process - eg BIST mode
resetSequence	1	Resets channel and all sensors on it
reframeSequence	2	Reframes channel
resetTx	3	Resets Transmit FIFO in Cypress transceiver
resetRx	4	Resets Receive FIFO in Cypress transceiver
loopBIST	5	Puts Cypress transceiver in BIST mode - requires external fibre loopback
setDiscardPolicy0	10	
setDiscardPolicy1	11	
setDiscardPolicy2	12	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

setDiscardPolicy3	13	
loadSerAddrReg	20	Load serial address register with chnlData[0]

typedef struct **T_CHNL_STAT** /* Channel Status returned by Hub */				
type	**Variable**	**Units**	**Bytes**	**Value**
ushort16	chanStatusRegister	bitmap	2	see section 3.5.15
ushort16	chnlRVSErrorCount	integer	2	count accumulated by MPC8260 in BIST mode
uint32	chnlCmd	enum	4	chnlCmd set to last cmd processed
			8	

typedef struct **T_HH_STAT** /* Hydra Hub Status */				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uchar	onOffLine	boolean	1	1 == on line
uchar	hubStatErrorCode	n/a	1	enum
int32	Encoder	count	4	
uchar	encA	n/a	1	Opto input 1 == HI == Fiber illuminated
uchar	encB	n/a	1	Opto input 1 == HI == Fiber illuminated
uchar	encZ	n/a	1	Opto input 1 == HI == Fiber illuminated
uchar	auxIn	n/a	1	Opto input 1 == HI == Fiber illuminated
uchar	auxOut	n/a	1	Opto output 1 == HI == Fiber illuminated
uchar	pad[1]		1	
uint32	resourceLoad	percent	4	**cpu/cmp/mem/mem**
			16	

typedef struct **T_FP_IDENT**				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
ushort16	fpType	integer	2	0..100 == Hydra Optimiser Type 101 = HRPV
ushort16	fpVersion	integer	2	1,2,3, etc
ushort16	sensorSktDiagnostic	integer	2	0x000 == not opened
ushort16	sensorSktProfile	integer	2	0x000 == not opened
			8	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

typedef struct **T_SENSOR_TYPE_IDENT**				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uchar	sensorType	integer	1	
ushort16	sensorVersion	integer	2	
ushort16	sensorSktDiagnostic	integer	2	0x000 == not opened
ushort16	sensorSktProfile	integer	2	0x000 == not opened
uchar	pad		1	
			8	

typedef struct **T_SENSORADDR**				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uchar	channelNum	n/a	1	Physical Hotlink channel 0..7
uchar	sensorNum	n/a	1	Sequential sensor number 0..
uchar	sensorType	enum	1	
uchar	detectorNum	n/a	1	
			4	

enum **msgErrorCode** /* Error Codes returned by Hub (any socket) */

size defaults to int == int32

Constant	**Value**	**Description**
none	0	
unrecognisedCmd	1	
invalidCmdOnline	2	
invalidCmdOffline	3	
invalidSensorType	4	
invalidSensorAddr	5	
invalidFP	6	
msgLenLong	7	
msgLenShort	8	
	9	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

5.3 Sensor Socket Commands

A socket connection is provided for each sensor type connected to the Hub. The sensor address (four bytes) is inserted prior to the DATA field. The following commands are common to all sensor types -

COMMAND	Hex	addr	DATA	RESPONSE	Comments
HS_RESET	0x01		-	hsIdent	
HS_REPORT_IDENT	0x02		-	hsIdent	

T_HS_IDENT hsIdent;
T_SENSORADDR addr;

typedef struct **T_HS_IDENT**				
type	**Variable**		**Bytes**	**Description/Notes**
uchar	logical address		1	
uchar	factorySerialNumMSB		1	
uchar	factorySerialNumLSB		1	
uchar	firmwarePartNumMSB		1	
uchar	firmwarePartNumLSB		1	
uchar	sensorTypeNumMSB		1	
uchar	pad[2]		2	
			8	

The following commands and data structures are specific to the HRPV sensor type

COMMAND	Hex	addr	DATA	RESPONSE	Comments
HS_HRPV_SET_STAT_CFG			hrpvStatCfg	hrpvStatCfg	
HS_HRPV_SEND_VIDEO			-	hrpvVideo	
HS_HRPV_REPORT_STAT_CFG			-	hrpvStatCfg	
HS_HRPV_ACCEPT_CAL_DATA			hrpvCalData	hrpvStatCfg	
HS_HRPV_REPORT_CAL_DATA			-	hrpvCalData	
HS_HRPV_SEND_EDGES			-	hrpvTransitions	

Where

T_HRPV_STAT_CFG **hrpvStatCfg;** /* see section 3.4 */
T_HRPV_VIDEO **hrpvVideo;**
T_HRPV_TRANSITIONS **hrpvTransitions;**

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

typedef struct T_HRPV_VIDEO				
type	**Variable**		**Bytes**	**Description/Notes**
T_SENSORADDR	addr		4	sanity check
uchar	video[pixNum]		1768	pixNum 0..1767
			1772	

typedef struct T_HRPV_TRANSITIONS				
type	**Variable**		**Bytes**	**Description/Notes**
T_SENSORADDR	addr		4	sanity check
uchar	transitionData		12..1020	3 bytes per transition – section 4.3.1 Doc 1765 includes sync pulse marker
			1024	

5.4 FP Socket Commands - HRPV

Each Frame Processor supports two socket connections - one dedicated to ONLINE operation and the other to OFFLINE.

The OFFLINE connection is provided for setup and diagnostics and provides access to all functionality except that output profile data is not acquired continuously - each line or block of data must be requested.

The ONLINE connection is the real time data pipe via which profiles are continuously made available to the requestor - typically a Decision Processor subsystem. In ONLINE mode, the only commands supported are HFP_SEND_NEXT_BUFFER and HFP_GO_OFFLINE requests.

COMMAND	Hex	DATA	RESPONSE	Comments
HFP_HRPV_RESET	0x01	-	hrpvFpStatus	
HFP_HRPV_REPORT_STATUS	0x02	-	hrpvFpStatus	
HFP_HRPV_GO_ONLINE	0x03	-	hrpvFpStatus	
HFP_HRPV_GO_OFFLINE	0x04	-	hrpvFpStatus	
HFP_HRPV_SET_FP_CFG	0x05	hrpvFpCfg	hrpvFpCfg	
HFP_HRPV_REPORT_CFG	0x06	-	hrpvFpCfg	
HFP_HRPV_RESET_FRAME_NUM	0x07	-	hrpvFpStatus	
HFP_HRPV_SEND_NEXT_SEGMENTS	0x10	-	hrpvFpSegments	
HFP_HRPV_SEND_NEXT_PERIMETER	0x11	-	hrpvFpPerimeter	
HFP_HRPV_SEND_NEXT_TIFF	0x20		hrpvFpTiff	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

any	0xFF	-	errorCode	

where

T_HRPV_FP_STAT hrpvFpStatus;
T_HRPV_FP_CFG hrpvFpCfg;
T_HRPV_FP_SEGS hrpvFpSegments;
T_HRPV_FP_PERIM hrpvFpPerimeter;
T_HRPV_FP_TIFF hrpvFpTiff;

typedef struct **T_HRPV_FP_STAT**

type	Variable	Units	Bytes	Description/Notes
uchar	onOffLine		1	1 == Online
uchar	pad[3]		3	
uint32	frameNum		4	current frame number
uint32	numBuffersRdy		4	
uint32	numBuffersOverwritten		4	
			12	

typedef struct **T_HRPV_FP_CFG**

type	Variable	Units	Bytes	Description/Notes
ushort16	numSensors	integer	2	
T_SENSORADDR	sensorAddress[8];	n/a	32	max 8 sensors per FP
ushort16	unitsOfProfile	n/a	2	1 = thousandths of an inch * 10 (Hydra) 2 = micron
int32	maxProfileX;	unitsOfProfile	4	
int32	minProfileX;	unitsOfProfile	4	
int32	maxProfileY;	unitsOfProfile	4	
int32	minProfileY;	unitsOfProfile	4	
uchar	Pad[4]	integer	4	
			56	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Segment data to the FP (CAE) is assembled as a 'block' of data, (type T_HRPV_FP_SEGS), which contains variable length lines of segment information for a board as detected between lugs, or just detected in a lugless system.

The term 'block' refers to either just the board data, all the lines between lugs, or tbd.

T_HRPV_FP_SEGS is of variable size, but this size can be calculated from the first 3 uint32 values sent from the Hub.

Segment data is sent as 16 bit run length (rle) values with the upper bit set to indicate the presence of wood. The rle's are written in pairs to a 32 bit (T_HRPV_SEG) structure with the second rle set to all 0's if it is not required.

The full hierarchy for a 'block' of data is as follows:

```
T_HRPV_FP_SEGS          {blockNum, numLines, numSegments)
    T_HRPV_LINE             {lineNum, numSegs, lineStat}
        T_HRPV_SEG              {rle[0], rle[1]}
                                {rle[0], rle[1]}
                                ..........
    T_HRPV_LINE             {lineNum, numSegs, lineStat}
        T_HRPV_SEG              {rle[0], rle[1]}
                                {rle[0], rle[1]}
                                ..........
    T_HRPV_LINE             {lineNum, numSegs, lineStat}
        T_HRPV_SEG              {rle[0], rle[1]}
                                {rle[0], rle[1]}
                                ..........
end
```

Note: T_HRPV_LINE can also be used by itself to send/receive mask data for lug locations etc.

typedef struct **T_HRPV_FP_SEGS**				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uint32	blockNum		4	block number for these lines
uint32	numLines		4	number of lines in this frame
uint32	numSegments		4	total number of segments in this frame
T_HRPV_LINE	line[numLines]		4..	
			12..	

The total size of the frame (uint32) = 3 + numLines + numSegments
(noting that each line requires a single uint32 for its lineNum etc, and T_HRPV_SEG segment is a 32 bit value, even though it contains 2 16 bit values, the second one of which may be set to 0's)

```
#define MAX_LINES_PER_FRAME     2000
```

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

typedef struct T_HRPV_LINE				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uint16	lineNum		2	line number for these segments
uchar	numSegs		1	number of segments in this line
uchar	lineStat		1	0 = OK, 1 etc = error
T_HRPV_SEG	seg[MAX_SEGS_PER_LINE]		4	
			4..	

```
#define MAX_SEGS_PER_LINE       32

struct T_HRPV_SEG
{
        T_SEGMENT   rle[2];
}

typedef uint16   T_SEGMENT; /* msb set == 1 == wood present (black), value == run length - unitsOfProfile*/
```

If there are no segments in the line, then numSegs = 0 and there is no data after lineStat;

If rle[1] is not required, then it is set to all 0's. (this will be true only if the board, or debris extends past the end of the sensors thus creating a single transition)

There is no segment data from the end of the board to the end of the sensor frame – for example if there is a board showing just two edges – left and right hand end – then this would be encoded as follows (8 bytes):

```
lineNum = count
numSegs = 2;
lineStat = 0;
seg[0].rle[0] = CCCC;        where CCCC is length of no board present with msb = 0
seg[0].rle[1] = BBBB;        where BBBB is board length with msb = 1;
```

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

typedef struct T_HRPV_FP_PERIM				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uint32	frameNum		4	frame number for these segments
uint32	numPoints		4	number of xy points in this frame
T_PERIM_PT	perimPt[0..numPoints-1]		4..	data points
			12..	

```
#define MAX_PERIM_PTS       8000

T_PERIM_PT   perimPt[MAX_PERIM_PTS];

typedef struct   T_PERIM_PT
{
        short16 x;
        short16 y;
}
```

typedef struct T_HRPV_FP_TIFF				
type	**Variable**	**Units**	**Bytes**	**Description/Notes**
uint32	frameNum		4	frame number for these segments
T_TIFF	tiffData		4	constructed from gapTiffMaker.h
uchar	pad[0..2]		0..3	pad bytes as necessary to mod 4
			12..	

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

6.0 Sensor Configurations / Performance

There are several sensors that could be used with the Hydra Hub. These are examined below in greater detail, but briefly these are:

HRPV – High Resolution Plan View sensor as proposed by 3DM based on Laser Caliper concept. These have a relatively modest data transfer requirement, but multiple heads. The Hydra Hub provides a well matched interface to these as many sensors can be daisy chained on to a single Hub channel. It is the choice of the system integrator how many sensors per channel depending on loop integrity concerns and need for other channels to service other sensors – eg LPS

High Speed Camera – under development by 3DM. The Hydra Hub is the primary interface to this device.

Existing Hydra Heads – This was the initial product for which the Hub was intended. It provides a cost effective interface to other Host systems with the intent of establishing applications in other markets for the Hydra product.

LPS2 sensors – CAE Newnes has expressed interest in integrating all sensors on a transverse frame with the Hydra Hub concept. Details of this are tbd but the concept should be workable.

6.1 HRPV

Per Hub

Worst case – 128 byte packets per sync

4 sensors
2 Khz scan rate
128 bytes per scan per sensor

Average data rate = 4 * 2Khz * 128 = 1 Mbyte /sec

Smart sensor – packed data – written if transitions found – sent when FIFO ½ full – Frame count read on each sync by FPGA

Typical 12" board
0.020" samples at 2 Khz (Board speed 0.020 * 2000 = 40 ips)
Length transitions – 12/0.020 per end = 600 -> 1200 total
Wavy edges – estimate 1" width – 6 transitions = 1"/0.020 * 6 = 300 -> 600 total
Total time per board = 12/40 = 0.3 seconds
Total transitions = 1200 + 600 = 1800
Encoding per transition – conservative 4 bytes
Burst data rate = 1800 * 4 / 0.3 -> 24 Kbytes/sec
Data per board = 1800 * 4 = 7.2 Kbytes

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

6.2 High Speed Camera

Per camera

Profiling
1000 lines
500 lps max
16 bit short == centroid

Data rate = 1000 * 500 * 2 bytes/sec
= 1 Mbyte/sec

Video
500 Mbytes/sec – not practical to send back to Hub in real time

6.3 Hydra Heads

6.3.1 General

6.3.2 Data Rates

The data rate calculations are based on the following parameters:

Pixels per line	768
Lines per field	246
Field rate	60 Hz
Centroid resolution	½ pixel
Centroid format	16 bit
Number of cameras per Head	2
Number of heads per hub	4
Calibrated data format (x,y)	32 bit (16 + 16)

6.3.2.1 Profiling

Per Head:

Centroids / sec = lines per sec

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

	=	Lines per field * Field Rate * Number cameras
	=	246 * 60 * 2
	=	29,520
Data rate	=	Centroids/sec * Centroid format
	=	29,520 * 16bit
	=	472,320 bits/sec
	=	59,040 bytes/sec

Per Hub:

Input centroid rate	=	Centroid rate/head * Number heads per hub
	=	29,520 * 4
	=	118,080
Output Cal data rate	=	Input Centroid rate * Cal data format
	=	118,080 * 32 bit
	=	3.8 Mbit/s

6.3.2.2 Video

Video is supported for diagnostic and calibration purposes, one camera at a time. It is impractical to consider real time video for measurement purposes with the current Hydra heads as there is one ADC per board and two cameras operating simultaneously. At best, a single camera could be considered but this would be a poor justification for supporting the required streaming data rates – not because of the raw data rate, but the changes required in the head firmware and receive FIFO's at the Hub.

The present video protocol is designed around the 8k capacity of the transmit and receive FIFO's. These FIFO capacities and support protocol will not change in the new Hub design.

6.3.3 Modifications Required

There are several changes required to the Hydra head to be compatible with the new Hub. These are itemised below:

6.3.3.1 Hardware

Change to plastic fibre optic cables

Replace HP transceiver – high speed transmit

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

low speed receive

Replace jumper cable and connectors with plastic

Replace Cypress Hotlink receiver with TL16C550C UART (receive only)

Remove receive FIFO

Remove isolated encoder interface, connector and all associated components

6.3.3.2 Firmware

Add command to uniquely address head based on serial number + Hub config

Modify command receive module to parse UART commands addressed to head

Add FPGA/firmware support for 'break' interrupt

These changes could be retrofitted to existing heads. The new head design can be built alongside the existing design without difficulty.

6.4 Transverse Frame Profiling – LPS2

Data consists of 16 bit range data taken at 1.5" intervals at 2Khz. For a 20' wide frame, the number of data points is then 20*12/1.5 = 160.

The input data rate is then 160 * 16 * 2KHz = 5.12 Mbits/sec for the top and bottom sections of the frame.

The output data rate is then 5.12 Mbits/sec in total for single 16 bit thickness data.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

Appendix I – MPC8260 VxWorks Boot Configuration

EST SBC8260 Boot

The SBC8260 was delivered with VxWorks boot code which supports the default network configuration as described in section 2.3.3 Networking the Host and Target, *Tornado 2.0 Users Guide*. The FTP server on the Host is **wftpd32.exe**.

Configuration

The Target and Host are configured as follows –

Target –

```
boot device          : motfcc
unit number          : 0
processor number     : 0
host name            : cadcam
file name            : vxworks
inet on ethernet (e) : 192.168.1.30
host inet (h)        : 192.168.1.1
user (u)             : vxworks
ftp password (pw)    : vxworks
flags (f)            : 0x8
target name (tn)     : ESTHub01
startup script (s)   : hubcode.txt
```

Host –

Wftpd32.exe is setup with a user account vxworks (in security settings) which sets c:\vxworks as the home directory. This directory is the location of any download files.

Boot Sequence

The SBC8260 boots in the following sequence

Boot VxWorks from Flash
    Initialises terminal and ethernet drivers
Load file **vxworks** from Host cadcam
    Installs telnet, shell and loader options
    Runs script file hubcode.txt
Hubcode.txt script
    Executes commands in this file – typically load *.o files and spawn tasks

A typical cold boot terminal display is shown below for the cold boot followed by VxWorks download –

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

ROM boot sequence

```
VxWorks System Boot


Copyright 1984-1998  Wind River Systems, Inc.

CPU: EST Corp. est8260 -- MPC8260 PowerQUICC II SBC
Version: 5.4
BSP version: 1.2/3
Creation date: Sep 20 1999, 17:06:15

Press any key to stop auto-boot...
 1
 0auto-booting...

boot device          : motfcc
unit number          : 0
processor number     : 0
host name            : cadcam
file name            : vxworks
inet on ethernet (e) : 192.168.1.30
host inet (h)        : 192.168.1.1
user (u)             : vxworks
ftp password (pw)    : vxworks
flags (f)            : 0x8
target name (tn)     : ESTHub01
startup script (s)   : hubcode.txt

Attached TCP/IP interface to motfcc0.
Subnet Mask: 0xffffff00
Attaching network interface lo0... done.
Loading... 873712
Starting at 0x100000...

àAttached TCP/IP interface to motfcc unit 0
Attaching network interface lo0... done.
NFS client support not included.
```

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained thereinfor other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing

Download VxWorks – Execute script

```
Adding 3185 symbols for standalone.

 ]]]
```

Downloaded VxWorks Image

The **vxworks** file downloaded from cadcam is built using the EST build options in Tornado 2. These are accessed from the pulldown build menu option.

The build name is **ESTWdbNetAppShell** The build uses the EST makefile in *Tornado2\target\config\est8260* which has an include file – rules.est – This file has been modified as noted below to add functionality to load a file from the Host and run a script.

Note the discussion in section 2.2.1 – Memory Map – and changes to various modules included in this build.

```
##############################################################################
#  Rule for creating a RAM downloadable vxWorks image with a
#  WDB Agent, over the network, with  EST app and Shell.
#  added INCLUDE_LOADER and INCLUDE_STARTUP_SCRIPT - jk Mar 2000
##############################################################################

.PHONY : ESTWdbNetAppShell.vxWorks save.ESTWdbNetAppShell

ESTWdbNetAppShell.vxWorks:
        make "EST_EXTRA_DEFINE = -DEST_TELNET_CODE -DINCLUDE_LOADER -DINCLUDE_STARTUP_SCRIPT"  \
        "EST_MACH_EXTRA = aEstDemo.o usrLib.o" vxWorks.st
ifeq ($(TOOLENV),68k)
        hex < vxWorks.st > vxWorks.hex
        $(EST_CONVERT) -s vxWorks_st.hex -b ESTWdbNetAppShell.bdx
        $(EST_CONVERT) -h vxWorks_st -c ESTWdbNetAppShell.ab -t m68k
else
        $(EST_CONVERT) -w vxWorks.st -b ESTWdbNetAppShell.bdx -c ESTWdbNetAppShell.ab -t ppc
endif
        $(CP) vxWorks.st ESTWdbNetAppShell.vxWorks

save.ESTWdbNetAppShell: ESTWdbNetAppShell.vxWorks
        $(CP) ESTWdbNetAppShell.bdx     prebuilt
        $(CP) ESTWdbNetAppShell.vxWorks prebuilt
        $(CP) ESTWdbNetAppShell.ab      prebuilt
        $(CP) ESTWdbNetAppShell.abx     prebuilt

##############################################################################
```

The output file vxworks.st is renamed vxworks and copied to cadcam\c:\vxworks

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for whic it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

3DM
Devices Inc

HYDRA Hub Host

SYSTEM DESIGN DESCRIPTION

May 2000

Prepared for:

3DM Devices

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

Revision History:

Version #	Date	Author	Description
1.0	May 10, 2000	ML	Gui description
1.1	May 12, 2000	ML	Architectural design
1.2	May 16, 2000	ML	Removed specific controller types
1.3	May 16, 2000	ML	Worked in the TAggregatorController
1.4	June 07, 2000	ML	Added design plan, etcetera
1.5	July 26, 2000	ML	General Revision

Referenced Documents:

Document	Source
1757 – Hydra Hub Design Description	3DM
1362 – Hydra Sensor Interface Detail Design Description	3DM
1085 – Technical Note – Hydra Hub Host Socket Implementation	3DM
1807 – Hydra Hub Initial Requirements Specification	3DM
1821 – Hydra Hub Simulator Design Description	3DM

Print Date: February 1, 2001 (10:06AM)

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

TABLE OF CONTENTS

1.0 Overview 3
1.1 Purpose 3
1.2 User Requirements 3
1.3 Development Tools 3
1.4 Scope 3
1.5 Intended Audience 3
1.6 Nomenclature 3
1.7 Design Objectives 4
1.7.1 Speed 4
1.7.2 Functionality 4
1.7.3 Expandability 4
1.7.4 Upgrade By Others 4
1.7.5 Robustness 5
1.7.6 Simplicity 5
2.0 Hub Subsystem 6
2.1 Basic Functionality 6
2.2 Architecture 7
2.3 Class Functionality 8
2.3.1 Buffers 8
2.3.2 Options and Persistence 8
2.3.3 Error Handling and Logging 8
2.3.4 Parameters 9
2.4 Socket Communication and Parameter Traffic 9
2.4.1 Hub Connection Example 9
2.4.2 Connect Sequence 10
2.5 Class Details 11
2.5.1 THubInterface 11
2.5.2 TParameterValue 11
2.5.3 TParameterSet 11
2.5.4 TParameterSetHost 11
2.5.5 TOptionsManager 11
2.5.6 TEventLogger 11
2.5.7 TCommunicator 11
2.5.8 TCommunicatorMessage 12
2.5.9 TBackendObject 12
2.5.10 TNamed 12
2.5.11 TController 12
2.5.12 TAggregatorController 12
2.5.13 TBackendObjectAggregator 12
2.5.14 TSensor 13
2.5.15 TDetector 13
2.5.16 THydraSensor 13

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

on as a reply to a command.

of a set of detectors.

onse to a previously sent message

Hub Host.

ler the Data Rate Calculations under

his is handled in the following way:
set once any data packet comes in.
onse message.

certain amount of time (as set by
again. If it still can't get the whole
o prevent reading a byte at a time.
ount of time (as specified by

ver possible. This is handled by the
p, and provides access to these upon
notes about these buffers:
in such a way buffer copying can be

ations.
rt, and whenever the number of used

for a long time (as specified by
ed.

Hydra Hub hardware. Although the
ation does that and more. It lets the
ion will deal only with the HRPV, and
sensor. However this application is

processors can easily be hooked into
lso, 32-bit integers are used to store
t expandability in the future.

eone new to the system to come to
vith comments, as well the complex

document or of any of the information contained
evices Inc. may otherwise agree to in writing.

1 13
..... 14
..... 14
..... 14
..... 14
..... 14
unicator 14
sor 14
sorCommunicator 14
ocessorCommunicator 14
sorGroup 14
icator 14
..... 14

..... 16

..... 16
nent> 16
t<tElement> 16
ey, tElement> 16
ted 16
..... 16
..... 16
ception 16

..... 17

end 17

..... 18

..... 19

..... 20
uent Changes 20
ter Changes to Window Displays 21
d Logging 21

..... 23

..... 25

proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained
urpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1.0 Overview

1.1 Purpose

The purpose of this document is to describe the system design of the Hydra Hub Host.

1.2 User Requirements

- 500MHz PIII or better
- 24 bit color or better
- 1024x768 display or better
- 2-button mouse
- Windows NT 4.0

1.3 Development Tools

- Borland C++Builder 5
- Marley THyperGrid 2.0, from www.pablop.demon.co.uk
- ImageMan Dll Suite 6, from www.data-tech.com

1.4 Scope

This document covers the graphical user interface (or gui) of the Hydra Hub Host application, as well as its internal system architecture. It also outlines some of the terminology used and the rationale behind the implementation.

1.5 Intended Audience

This document is intended for internal review at 3DM Devices and comment from outside parties. All use of this document is subject to the terms of non-disclosure agreements.

1.6 Nomenclature

backend	Refers to the subsystems excluding the gui and the utility classes. The Hub Subsystem and the Display Subsystem are examples.
channel	A physical serial interface—one of 4 or 8—to which sensors are connected, one or several per channel, one sensor type per channel.
command (message)	A message sent out from the application to the hub via TCP/IP, requesting data and/or requesting an action to be performed.
descriptor	A structure used to identify exactly which component in the Hub Subsystem a parameter applies to.
detector	A device mounted on a sensor to do the actual detecting. There can be multiple detectors per sensor. For example, a Hydra Sensor has two detectors (or cameras).
hub parameter	An attribute of the THydraInterface. It is identified by an id in the form of HP_XXX.
lazy parameter	A hub parameter whose set calls are ignored until a given laziness interval passes since the most recent set, at which point the parameter's value is actually set.
laziness interval	The time the THydraInterface will wait before committing a lazy parameter's value.
option	A parameter that is persisted via the registry. Not all parameters are necessarily options.
parameter	A property or attribute, whose value is of a variant type.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

response message	A message coming back from the hub to the application as a reply to a command.
sensor	A device (for example, a Hydra Sensor) that consists of a set of detectors.
timeout	When a message from the hub is not received in response to a previously sent message after a certain amount of time.

1.7 Design Objectives

The following are considered to be the key design objectives for the Hydra Hub Host.

1.7.1 Speed

- The application must be able to handle 3.8 Mbit/s, as outlined under the Data Rate Calculations under Profiling in document 1757.
- The biggest potential bottleneck is in the socket communications. This is handled in the following way:
  - TCommunicator's thread waits on a ::select() signal, which is set once any data packet comes in.
  - It reads the header, and determines the total size of the response message.
  - It tries to read the entire message.
  - If it can't get the entire message at once, it waits a certain amount of time (as set by COMMUNICATOR_RECEIVE_SLEEP_AMOUNT) and tries again. If it still can't get the whole thing, it doubles this wait time and waits again. This is done to prevent reading a byte at a time.
  - If it doesn't read this message in a certain amount of time (as specified by COMMUNICATOR_RECEIVE_TIMEOUT), it times out.
- Another important issue is to avoid copying data buffers whenever possible. This is handled by the Buffer Server. This object allocates a number of buffers off the heap, and provides access to these upon request. They are used to store TCommunicator messages. Some notes about these buffers:
  - They can be passed from object to object by buffer ids, and in such a way buffer copying can be avoided.
  - Returned buffers are reused, avoiding extraneous heap allocations.
  - A certain number of buffers are allocated off the heap to start, and whenever the number of used buffers approaches this number, more are allocated.
  - If there are a lot of buffers allocated and not used for a long time (as specified by BUFFER_SERVER_STAY_BEHIND), the memory will be freed.

1.7.2 Functionality

The purpose of this application is to provide a front end interface for the Hydra Hub hardware. Although the previous Hydra application was intended solely for calibration, this application does that and more. It lets the user view scan data as it's coming through the HRPV. Initially, this application will deal only with the HRPV, and provide visualization of this data coming in and calibration of the HRPV sensor. However this application is designed so that in the future, it can also run the hydra and other sensors.

1.7.3 Expandability

The Hydra Hub Host is written so that new sensors, detectors, and frame processors can easily be hooked into the system. Linked lists are used extensively to avoid hardcoding limits. Also, 32-bit integers are used to store values, even when their corresponding data members are smaller, to permit expandability in the future.

1.7.4 Upgrade By Others

This documentation is provided to make this as easy as possible for someone new to the system to come to speed with its design and function. Every class function is documented with comments, as well the complex parts of code.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

1.7.5 Robustness

The Hydra Hub Host must be able to handle any error conditions that may arise. Error logging is provided for error, warning, information, and bug messages. The NT event log is used to store this information, and it is also displayed inside a window in the application.

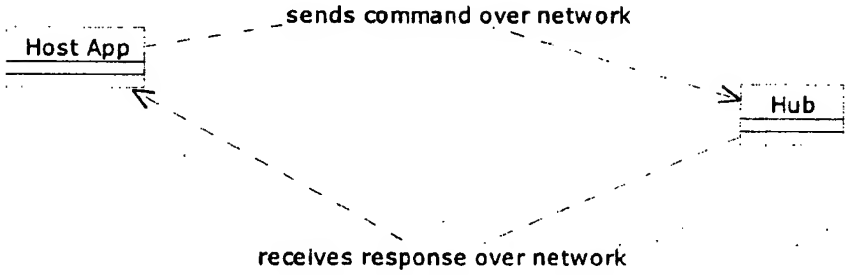
1.7.6 Simplicity

The system architecture is designed such that there are no unnecessary classes.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

2.0 Hub Subsystem

2.1 Basic Functionality



The Hydra Hub Host application connects to the Hub over the network via TCP/IP, and issues commands. It can:

- ask for hub status (which is done constantly through a timer)
- ask for the status of any sensor attached
- get RLE data
- get TIFF data from a frame processor
- get calibration data
- set calibration data

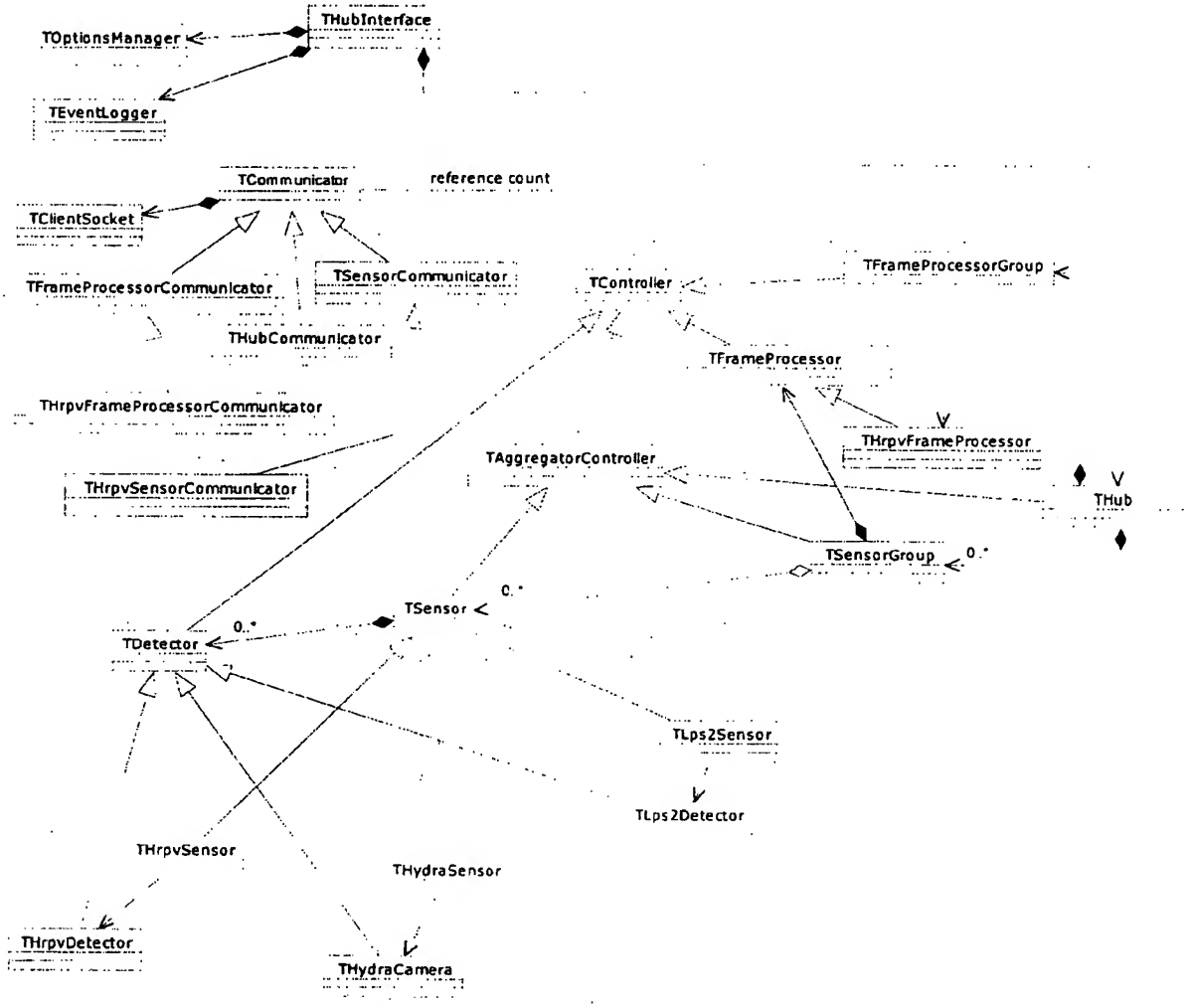
All messages are interlocked per socket; every time a command is sent out from the Host app to the Hub, no further commands are sent out on that socket until a response comes back.

Thus, the application can be used for testing and calibration of sensors. Also, it can be used in the field for diagnosis of sensor problems, as data can be displayed through the app as well.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

2.2 Architecture

The following UML diagram illustrates the architectural design for the hub subsystem. Triangular arrows denote derivation. Diamond arrows denote aggregation, where solid diamond arrows represent ownership.



This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

2.3 Class Functionality

This section gives an broader overall description of the various classes by their function. The following subsections are in no particular order.

2.3.1 Buffers

Handling of data is an important issue in this system, as it is crucial to do as little data copying as possible to maximize efficiency. A single global instance of TBufferServer provides these buffers, which are 64KB each. It allocates a number of them off the heap, and allows users to "take" (or lock) these buffers. Once the users of TBufferServer "return" (or unlock) these buffers, they are put back into its pool of available buffers. This way dynamic memory allocations and deallocations are kept at a minimum. If there are ever any buffers not available, the buffer server goes and allocates some extras just in case. These are freed once they have been left unused for a certain amount of time (as specified by BUFFER_SERVER_STAY_BEHIND).

2.3.2 Options and Persistence

Options are a collection of parameters that have been specified for persistence by TBackendObject derivatives. Upon construction, the TBackendObject derivative call AddOptionsParameter(), passing the parameter id of the parameter which is to be persisted. This information will later be used by the options manager.

Upon construction, the TOptionsManager singleton creates TOptionsSubkey objects for each main TBackendObject (ie for the THubInterface, representing the THub, and the TDisplayManagerInterface, representing the TDisplayManager). The options manager supports four commands. Retrieve() iterates through the list of TOptionsSubkey objects and retrieves their option parameters from their parameter sets. The Save() function takes the current options in the options manager and writes them to the registry. The Load() function iterates through all the options in the options manager and loads them from the registry. And finally, Apply() takes the current set of options and writes them in the parameter sets of the applicable TBackendObjects.

For display objects, the process is a little different, as windows are not connected to their parameter set. Whenever a parameter changes in the TDisplay's parameter set, a function in the window class is called through a pointer. This function is called the Apply Handler. It goes through a switch statement and updates the appropriate window property based on the parameter value passed to it from the TDisplay's parameter set.

2.3.3 Error Handling and Logging

The root of all error handling and logging is the TEventLogger singleton class. This class supports a set of functions such as ErrorMessage(), WarningMessage(), and InfoMessage(), each of which deal with a separate type of event. Whenever one of these functions are called, the event is logged into the NT event log, and a message is sent to all window observers. TEventLogger derives from TWindowSubject, which allows windows to registers themselves as observers of the class and receive notification messages when new events occur. This registration is done through THubInterface::AddEventLoggerObserver() and THubInterface::RemoveEventLoggerObserver().

One of these observers is the main form, which changes the states of a few leds in the bottom left corner to show that there are new events in the event log. The user can then click on the log button in the main tool bar to display the log window. Furthermore, error messages pop up the log window automatically.

TBackendObjects derive from TEventLoggerHost, which provides member functions for ErrorMessage(), WarningMessage(), and InfoMessage(), that simply delegate to the TEventLogger singleton. These should not be called directly in most cases, as macros labeled INFO_MESSAGE(), ERROR_MESSAGE(), etcetera, are provided. These allow for more verbose debugging information that is supplied in hidden macro parameters.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

2.3.4 Parameters

The Hydra Hub Host, since it deals with the hub through socket calls, must deal with a lot of messages, most of which are simply accessors for various parameters of the hub. Since there is a large set of these, and always room for expansion, a sound and flexible design is required.

Every parameter is given a unique identifier. This is called a parameter id. All parameter ids are in the form HP_XXX (HP for Hub Parameter). You can find them in ParameterDef.h. They are further distinguished in terms of what component they refer to, as follows:

HP_SYS_XXX	The entire application.
HP_HUB_XXX	The hub itself.
HP_FPG_XXX	Frame processor group.
HP_FP_XXX	Frame processor.
HP_SSG_XXX	A sensor group.
HP_SSR_XXX	A sensor.
HP_DET_XXX	A detector.
HP_DIS_XXX	The display.
HP_DISX_XXX	A custom display parameter.
HP_MSG_XXX	A communicator message.

Descriptors are used to indicate to describe the particular component that is being referred to. For example, a TSensorDescriptor is used to indicate which sensor and which detector applies when setting a HP_SSR_THRESHOLD parameter.

TDescriptor is used as a base class for all of the different types of descriptors. Once a parameter is identified as referring to a given set of components, via a dynamic_cast, it can be routed to the specific component using the proper descriptor derivative.

2.4 Socket Communication and Parameter Traffic

The TCommunicator class is the engine for message communication. It aggregates a TPowerSocket, and therefore there is a socket open for ever instance of TCommunicator. Each TController object points to a communicator, although it does not necessarily own the object, as there can be many controllers to a single communicator. For example, there multiple sensors of the same type share the same TCommunicator object, which is owned by their TSensorGroup.

2.4.1 Hub Connection Example

Message traffic is structure the same way for most messages, and so this example is a good start at explaining how things work. The sequence is as follows:

- The user clicks the connect button.
- The HP_HUB_CONNECTED parameter is requested to be set to true by the gui.
- This is delegated to THub by the THubInterface, using the passed descriptor.
- THub's TParameterSetHost overridden RequestSetParameterValue function is called to request changing of this parameter value.
- THub calls it's communicator's Connect() function.
- TCommunicator initializes its socket thread and opens and connects to its socket. If this fails, an error is logged and the process fails.
- THubCommunicator creates a new TCommunicatorMessage and sets its command id to HH_REPORT_STATUS. It also sets the timeout to 2 seconds, as this particular message usually takes longer.
- TCommunicator's SendMessage() function is called. It adds the new message to its command queue and triggers a new command event.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

- THubCommunicator returns, and since this was done on the main ui thread, functionality is returned to the user.
- Meanwhile, TCommunicator's socket thread wakes upon an event being signaled that there is a new command in the queue.
- A sequence id for this message is generated from the set of currently available sequence ids.
- This message is sent over the socket, and if successfully sent, is put into the pending message queue. The purpose of this queue is to keep track of the messages that have been sent out, and to log an error if an expected response times out.
- Eventually, the socket thread receives a reply from the hub, a status report message.
- TCommunicator places this reply message into a response message queue and removes its associated message from the pending message queue (assuming their sequence ids match).
- An event is signaled to the controller that sent this command (in this case it would be the THub object).
- THub acts on this event and checks the communicator's response message queue with the use of PeekMessage().
- Once it is satisfied that this message applies to itself, it calls PopMessage(), which removes the message off the communicator's queue, and takes ownership of this message.
- It determines that it is a HH_REPORT_STATUS message and sets the HP_HUB_CONNECT parameter value accordingly. It wraps the call to SetParameterValue() between calls to BeginForceAcceptRequest() and EndForceAcceptRequest(), which prevents a call to RequestSetParameterValue(), therefore preventing an endless cycle.
- THub's apply handler is called (this is the function called by TParameterSetHost after parameter values change).
- This apply handler is delegated to THubInterface, which sends out a UM_NEW_PARAMETER_UPDATES message to the gui.
- The gui handles this message and pops TUpdateInfo objects from the THubInterface.
- Every one of these objects contains a parameter id which is used to retrieve the parameter's value from the THubInterface and then update the appropriate control (which in this case is the connected / disconnected led at the bottom right of the status bar).

On the other hand, TSensor and TDetector objects share the TSensorCommunicator created by their parent TSensorGroup object. Every controller (TSensorGroup, TSensor, and TDetector are all controllers) waits until its communicator's message event is signaled. Then, the controller checks the message queue and pops off the queue the response to the command it sent and acts on it.

From the gui end of things, if a set or get comes in for a detector, for example, the THubInterface passes this request to the hub object, which passes it to the sensor group, to the sensor, and finally to the detector. All this passing is done blindly with the help of TAggregatorController functions, which all these (except TDetector) are derived from.

2.4.2 Connect Sequence

- An HH_REPORT_STATUS command is sent out. If and when the response is received, the hub is connected.
- Once connected, an HH_REPORT_CFG command is sent to the Hub. Once the message comes back, the appropriate parameters are sent and the appropriate TController objects are created according to the configuration. It is worthwhile to note that the HH_REPORT_CFG command returns the type of sensor and count for each channel, but every TSensorGroup contains sensors of one type only, and there is only one TSensorGroup per sensor type. This means that two channels can have sensors of the same type, but their combined sensors will be represented by a single TSensorGroup object.
- A HH_GET_FP_IDENT command is sent out for each frame processor. A response is received which indicates the frame processor type and version, and also the port number for that frame processor. At that point, the frame processor object is created.
- Once a frame processor object is created, a HH_LAUNCH_FP command is sent out. A T_HH_STAT is returned, indicating success if the hubStatErrorCode is EC_NONE. Upon a successful response, the TFrameProcessor connects to a new socket via its communicator object.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

- For every sensor group, a socket must be opened. This is done by first sending out a HH_GET_SENSOR_TYPE_IDENT, whose response identifies the port which is to be used. Upon success, a HH_LAUNCH_SENSOR_TYPE command is sent out to launch the sensor socket.

2.5 Class Details

2.5.1 THubInterface

This class separates the gui from the backend components. It acts as a façade, allowing the gui to be oblivious to any components under the THubInterface. Every functionality required by the gui is supported through this class, which then delegates to the appropriate objects beneath it.

In order to persist settings, it owns an instance of TOptionsManager. In order to log messages and display errors to the user, an instance of TEventLogger is owned. An instance of THub is also needed to maintain hub properties, and to manage the set of sensor groups, sensors, and detectors.

2.5.2 TParameterValue

This class is similar to the VARIANT type, in that it represents a value that can vary in its type. However it is more user-friendly in that it provides overloaded constructors and operators for all its supporting types, which are int, bool, double, and String. The actual value is always stored as a string.

2.5.3 TParameterSet

This class represents a set of parameters, indexed by their ids. Ideally, this class will be implemented with the use of a hash table, but we still as yet haven't found a nice one for C++Builder. Here, parameter values can be set and retrieved through accessor functions that take parameter ids as inputs.

In addition, parameters can also have their own parameter sets. This is so that the user can flag parameters with attributes such as read-only, etcetera. These internal parameter sets are created only upon request, so they do not waste any additional memory when unused.

2.5.4 TParameterSetHost

This class aggregates a TParameterSet upon construction, and deletes it upon destruction. It is a base class that provides protected access to this aggregate.

2.5.5 TOptionsManager

This class is used by the THubInterface to read and write user settings for the application. It abstracts the THubInterface from having to deal with the registry. This component can easily be rewritten to read and write these settings to an alternate location, such as a database.

2.5.6 TEventLogger

This class traps all error and warning messages. It can be run in either silent mode, which just logs the messages, or in loud mode, in which in addition to logging the messages, the user is immediately notified via dialog boxes. Logging is done to the Windows NT Event Viewer.

2.5.7 TCommunicator

This class wraps the generic functionality of hydra hardware messages, and provides a set of functions to communicate with this hardware while abstracting the user from needing to know the details of dealing with the underlying socket calls. It maintains a message queue of all response and command messages. Whenever a new response comes into the queue, it notifies its associated TController object. TCommunicator derives from TReferenceCounted, to implement reference counting so that it is deleted only when the last person who is pointing to it calls Delete().

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

TCommunicator aggregates a TSmartThread object in order to handle socket communications. For these communications it uses TPowerSocket.

2.5.8 TCommunicatorMessage

This class wraps a message. This message includes the common elements such as a sequence id, a command id, and body data. The message data is stored in a buffer supplied by the buffer server. The message also allows the user to associate another message with it. This is used to supply the message that was sent out with the message that came in, once message processing occurs. Also, the TCommunicatorMessage class derives from TParameterHost, and thus the user can set various HP_MSG_XXX parameters to affect behavior once the message is processed.

2.5.9 TBackendObject

A base class for objects of the backend that hold a set of properties and can be accessed via TDescriptor objects. This class derives from TEventLoggerHost, allowing for message logging and naming through TNamed. It also derives from TParameterSetHost in order to be able to host a set of parameters.

2.5.10 TNamed

Deriving from this simple class allows an object to name itself. TNamed maintains a String holding this name, and provides it via the GetName() function.

2.5.11 TController

This class contains a thread that waits until a response message comes in through its associated TCommunicator object. It then calls a virtual function to handle this new message, and removes it from the queue. Any commands can be sent by accessing the associated TCommunicator object directly.

The reasoning behind using a separate TController class is to decouple what action is performed upon receiving a message (the TController's task) from the encoding, decoding, sending, and receiving of these message (the TCommunicator's task).

The TCommunicator and TController work as a pair, although usually the TController owns the TCommunicator.

TController provides handling functionality of connection parameters. This is done through SetConnectParameters(), passing it the parameter ids which are to be handled. After these are supplied, TController will automatically try to connect the communicator object when the connected parameter is set to true, and will disconnect when it is set to false. The socket port and ip address parameters will also be handled automatically.

2.5.12 TAggregatorController

This class derives from the template class TBackendObjectAggregator and represents a controller that maintains a set of aggregates, and automatically delegates the proper set and gets that come with descriptors to the proper aggregate controllers.

2.5.13 TBackendObjectAggregator

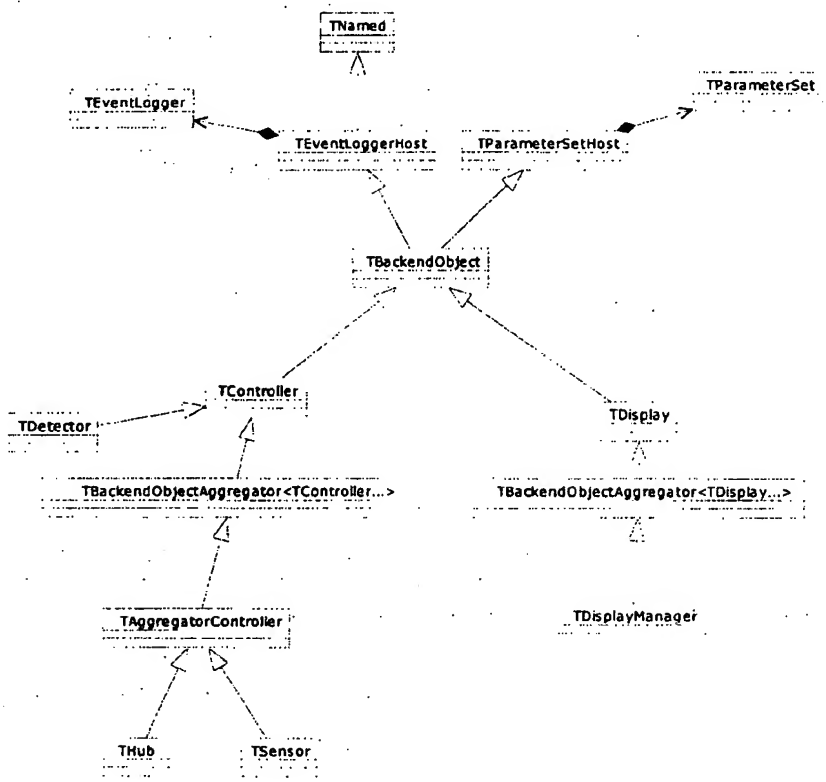
This is a template base class that abstracts the functionality of an aggregator of tAggregate objects but also derives from tAggregate itself. A tangible example of this is the TSensorGroup, which itself is a TController, and has a collection of TSensor aggregates, each of which are themselves TController derived.

This class takes three template arguments: class tAggregate, class tDescriptor, and class tConstructorInfo. The first specifies the aggregate type, from which both the aggregator and its aggregates are derived. The second specifies the descriptor type used for parameter setting and getting. The third parameter specifies the type to be passed to the tAggregate base class of the TBackendObjectAggregator.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

When setting and getting parameters, the class must know which member in the tDescriptor class it must access. This member is used to determine which aggregate is being referred to. This member is specified through the constructor of TBackendObjectAggregator, which takes in a pointer to a tDescriptor class data member of type int.

The following diagram illustrates these relationships in more detail.



2.5.14 TSensor

This class represents a sensor, and maintains all its attributes. It also owns a collection of TDetector derivatives, to represent the set of detectors for this sensor.

This class also derives from TAggregatorController, which delegates gets and sets to the proper TDetector aggregates. It is passed a TSensorCommunicator which it uses, increasing its reference count.

2.5.15 TDetector

This class represents a detector, which is a part of a sensor. It holds properties unique to detectors. It also derives from TController.

2.5.16 THydraSensor

This class is derived from TSensor, and represents the attributes of a hydra sensor. Upon creation, it aggregates two instances of THydraCamera as its detectors.

2.5.17 THydraCamera

This class is derived from TDetector, and maintains the attributes of a hydra camera.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

2.5.18 THrpvSensor

This class maintains a set of attributes for the hrpv sensor. It only allows for either 3, 6, or 9 aggregates of THrpvDetector in its set of detectors.

2.5.19 THrpvDetector

This class maintains a set of attributes for the hrpv detector, which is used by the hrpv sensor.

2.5.20 TLps2Sensor

This class maintains a set of attributes for the lps2 sensor. It aggregates TLps2Detectors into its detector set.

2.5.21 TLps2Detector

This class maintains a set of attributes for the lps2 detector, which is used by the lps2 sensor.

2.5.22 TSensorGroup

This class maintains a set of sensors. They are grouped in such a way that they all share a common socket. The TSensorGroup class does this by instantiating a TSensorCommunicator, and passing this pointer to all the TSensor objects it creates.

2.5.23 TSensorCommunicator

This class derives from TCommunicator, and therefore acts as a socket wrapper. Its task is to communicate sensor related messages for a set of sensors.

2.5.24 TFrameProcessor

This class represents a frame processor, and maintains all its attributes.

This class also derives from TController, and determines what action is to be taken for messages dealing with frame processors. It instantiates a TFrameProcessorCommunicator to perform the socket work.

2.5.25 TFrameProcessorCommunicator

This class derives from TCommunicator, and therefore acts as a socket wrapper. Its task is to communicate frame processor related messages for a set of sensors

2.5.26 THrpvFrameProcessorCommunicator

This class derives from TFrameProcessorCommunicator, and implements communication functionality specific to the hrpv frame processor.

2.5.27 TFrameProcessorGroup

This class derives from TController, and maintains a set of frame processor objects. It is aggregated by the THub object and takes a pointer to the hub's THubCommunicator, which it uses to handle frame processor messages dealing with the creation and destruction of TFrameProcessor objects.

2.5.28 THubCommunicator

This class derives from TCommunicator, and therefore acts as a socket wrapper. Its task is to communicate messages dealing with the hub itself or the entire system as a whole.

2.5.29 THub

This class represents a hub, and maintains all its attributes. It maintains a list of TSensorGroup objects. It aggregates a TFrameProcessorGroup to handle frame processor related functionality.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

This class also derives from TAggregatorController which delegates get and sets to the proper TSensor aggregates. THub determines what action is to be taken for messages dealing with the hub itself. It instantiates a THubCommunicator to perform the socket work, and passes it to its base class TController.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

3.0 Utility Subsystem

There are several utility classes in use. In order to not get too dependent on Borland libraries, wrappers have been built. These wrappers should be used in all cases.

3.1 Classes

3.1.1 TTypedList<tElement>

This class maintains a linked list of tElement objects allocated off the head. When these are passed in, the list object takes ownership, and deletes them off the heap upon destruction and where appropriate. Internally, this is based on TList.

3.1.2 TSimpleTypedList<tElement>

This class maintains a linked list of tElement objects stored by value. When these are passed in, the list object makes a copy. This is great for storing simple types. The type must be castable to a void*.

3.1.3 TTypedTable<tKey, tElement>

This class maintains a hash table of tElement objects allocated off the heap, and index by the use of tKey objects. When these tElement objects are passed in, the list object takes ownership, and deletes them off the heap upon destruction and where appropriate. The tKey type is usually type int, but any type that supports the modulo operator %, or has that operator overloaded, can be used as a key type.

3.1.4 TReferenceCounted

This is a base class that implements reference counting. It provides two functions, Multiply(), which adds a reference to the object, and Delete(), which decrements the reference count and calls the delete operator if appropriate.

3.1.5 TSmartThread

In order to get around the problem that VCL doesn't support multiple inheritance, TSmartThread derives from the TThread and implements a thread class that can be aggregated rather than derived from. It also provides a thread event that is signaled upon termination.

3.1.6 TPowerSocket

This class encapsulates the Berkley socket api. It was the original intent to use C++Builder's TSocket class, but they were found to be inappropriate (see Technical Note 1805). For error handling, this class throws TPowerSocketException objects, which the user of TPowerSocket can then trap.

3.1.7 TPowerSocketException

This class represents a socket exception thrown by TPowerSocket. It stores the error id (one of the WSAEXXX error constants), and provides the user with access to this and a string message describing the error.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc may otherwise agree to in writing.

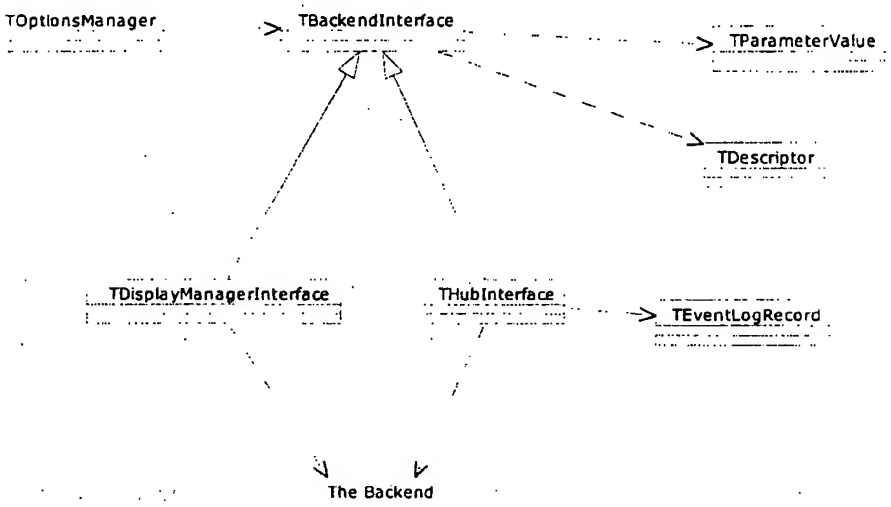
4.0 Gui Subsystem

The graphical user interface will consist of a resizable form broken up into three separates panes with the help of splitters. These panes can be resized by moving the splitters about, just as expected in other windows applications. (Note that these callouts won't print properly unless you turn on Rasterize Graphics in the printer options).

4.1 Talking to the Backend

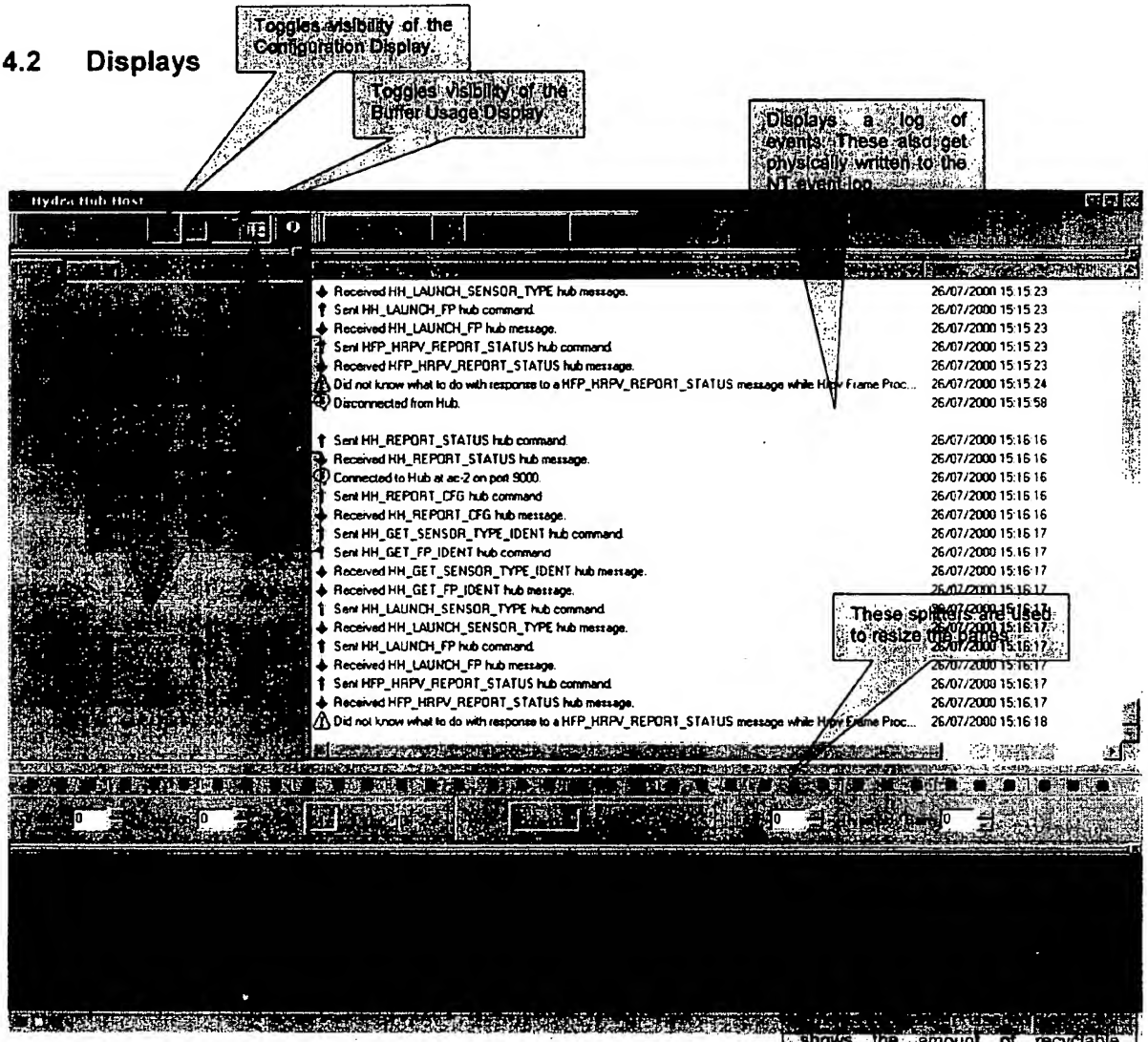
This is done through a number of interface objects that abstract the gui from having to talk to all the inner backend classes. TBackendInterface is supplied primarily for TOptionsManager's benefit, so that it can generalize between a TDisplayManagerInterface and a THubInterface. TDescriptor is used to describe backend components via ids. TParameterValue is a "polymorphic" type similar to OLE_VARIANT, and is used to get and set parameter values. TOptionsManager is used to apply and retrieve settings to and from the system, and also to read and write these settings to and from the registry.

The following diagram shows the classes that the gui must know about in order to access the backend.



This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

4.2 Displays



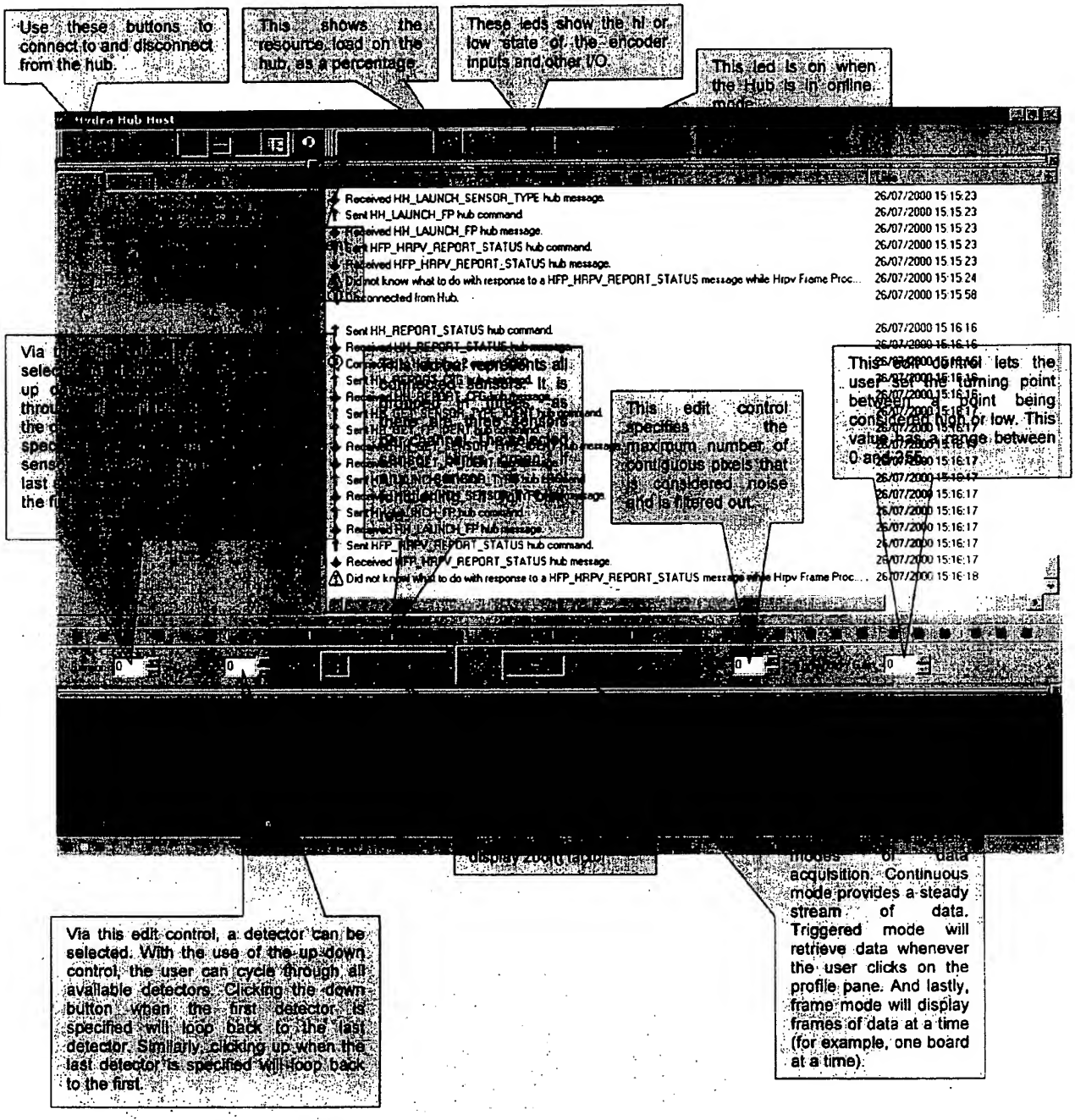
Not shown here are:

- A profile view of the objects being scanned.
- A oscilloscope showing the selected sensor data.
- A property page grid that will allow the user to calibrate the hub.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

4.3 Controls

This section describes the various controls found in the various toolbars, panes, etcetera (HRPV specific).



This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.4 Updates

4.4.1 Dealing with Frequent Changes

Changing the value of certain controls requires a lengthy socket transfer. This produces a problem with such controls as up down spin controls, which the user must click multiple times to change the value from the original setting to the desired setting. For example, what we don't want is to do ten socket transfers when the user changes the filter from 0 to 10.

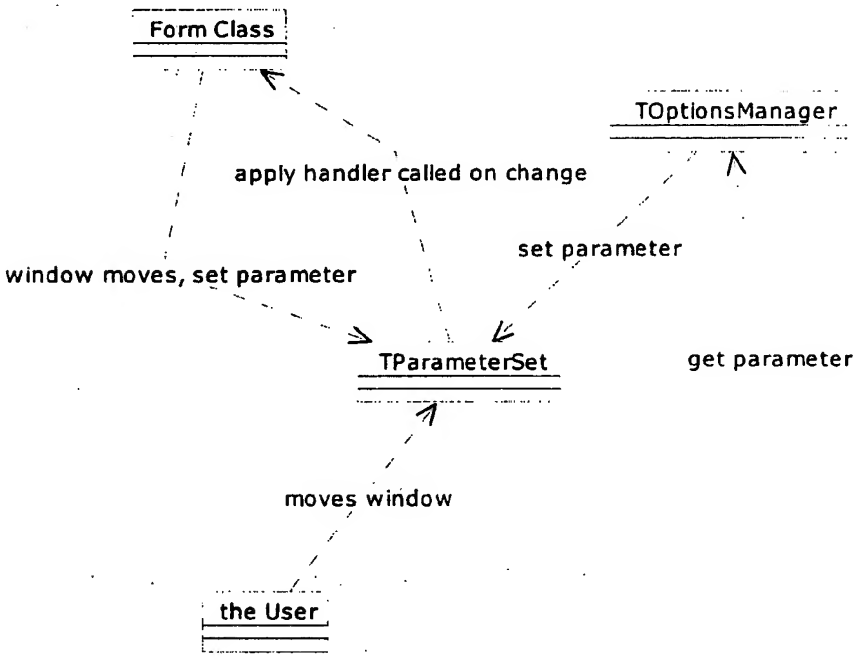
The solution to this problem is to wait a certain amount of time—say, a second—from the last modification of the setting to the actual updating of the value in the system. THubInterface accomplishes exactly this. It provides accessor functions for getting and setting various system values. The interval before the value is set is called the laziness interval. Here is the process which a user of THubInterface should go through:

1. Call THubInterface::RequestSetParameterValue() each time a control value changes in the gui.
2. Catch any UM_PARAMETER_CHANGED messages.
3. When this message comes in, call THubInterface::PopUpdate() to retrieve the parameterId and the descriptor for the parameter implicated in the latest update.
4. Retrieve the value of this parameter through THubInterface::GetParameterValue().
5. Update the control.

See the section on Parameters in the Hub Subsystem for more details.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

4.4.2 Applying Parameter Changes to Window Displays



The above figure illustrates how display manager parameters interact with the gui and the options manager. This scheme is required so that the gui layout can stored and written to the registry upon shutdown, and conversely, so that it can be loaded and restored upon startup.

Whenever the user manipulates a TDisplay-associated window, this is trapped via the event handler. For most of the basic window resizing and movement, the associated TDisplay object hooks into the window's event map, and for any additional parameters that cannot be generalized to all displays, these are handled by the form's class. These event handlers set the appropriate parameter in the parameter set for that display. This keeps the parameter values in sync with the properties of the actual windows, and allows us to save these values upon shutdown.

Now when we start up the application, we retrieve these values from the registry and call the option manager's Apply(). This triggers a function pointer called the ApplyHandler (which is a feature of the parameter set host) to be called. The ApplyHandler takes the parameter values and sets the window properties to reflect them. And in such a way, the display is restored.

Is there a circular update here? Yes, but that ends once the windows settle down in their correct positions, as ApplyHandler is called only when the value is change—not whenever it is set.

4.4.3 Error Handling and Logging

Error handling is done through a class called TEventLogger. This class supports member functions ErrorMessage(), WarningMessage(), and InfoMessage(), each of which logs a different type of event. TEventLogger implements logging with the help of CEventLog, whose sole purpose is to write into the event log. TEventLogger may be eventually extended to produce message dialogs to the user.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

TEventLogger is not usually used directly. The proper way to log events is to derive from TEventLoggerHost. This class provides member functions that delegate to the TEventLogger directly, and also provide for naming of the object, so that this information can be included within events.

There are times when code needs to check for validity of pointers and other safety checks, which should never fail upon release of the software. This is usually done through the use of assert(), but unfortunately, this function also exits the application, and never provides any error feedback once the software is released. The header file Bug.h contains a macro labeled BUG() which allows the caller to indicate a bug has occurred. This pops up a message to the user, indicating a bug has occurred, and specifies the source file name and line number, and also logs the error event.

The TEventLogger derives from TWindowSubject, which allows the gui to update its display on response to new events. This is how the event led bar is updated.

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

5.0 Development Plan

Build	Elements	Coded and Functional	Testing and Comments
Build 0			
Goals	Gui controls functional	Yes	
	Options persist	Yes	
	Events are logged properly	Yes	
	Connection can be made to the hub	Yes	
Testing	Play with gui controls such as spin controls, etcetera		Yes
	Check tab order		Yes
	Check docking and undocking works properly		Yes
	Check persistence of supported options		Yes
	Check window position, state, and dock state persistence		Yes
	Delete 3dm registry key and run system		Yes
	Disconnect network cable and try to connect to the hub		Yes
	Try running more than one instance of the app (should not let you)		Yes
Build 1			
Goals	Can retrieve hub configuration	Yes	
	Can retrieve sensor and frame processor configuration	Yes	
	Open other sockets besides the main hub socket	Yes	
	Implement hub socket commands and responses (except channel control commands)	Yes	
Testing	Connect with 0 frame processors.		Yes
	Connect with 1 frame processor.		Yes
	Connect with maximum amount of frame processors allowed by Simulator.		Yes
	Click connect button while already connected. Should not have any effect.		Yes
	Click disconnected button while already disconnected. Should not have any effect.		Yes
	Click reboot button when connected. Should reboot properly.		Yes
	Try hitting reboot continuously, without waiting for the system to carry out the action. Everything should still work, or fail gracefully.		Yes
	Try hitting the connect, disconnect, and reboot buttons in a random fashion, without waiting for the system to carry out the actions. Everything should still work, or fail gracefully.		Yes
	Always watch the Buffer Usage display and make sure all buffers eventually get freed and the display eventually shows all green.		Yes
	Vary the number of hrpv sensors connected in the simulator and make sure the correct configuration is displayed in the Logical View of the Configuration display in the host.		Yes
Build 2			
Goals	Implement sensor socket commands (except hrpv cal data commands)		
	Scope data and sensor configuration		
	NT detection.		
Testing	Check that scope data is as expected		
	Resize form and make sure scope gui controls display properly		
	Try running the app on a non NT machine. It should fail gracefully with an informative message to the user.		
Build 3			
Goals	Frame processor socket commands		
	Implement hub channel control commands		
	Retrieve tiff image		
Testing	Resize form and make sure scope gui controls display properly		
	Ensure frame data is accurate		
	Ensure the system can keep up with incoming data in continuous mode		
Build 4			
Goals	Calibration supported		
	Options dialog is complete		
Build 5			

This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing

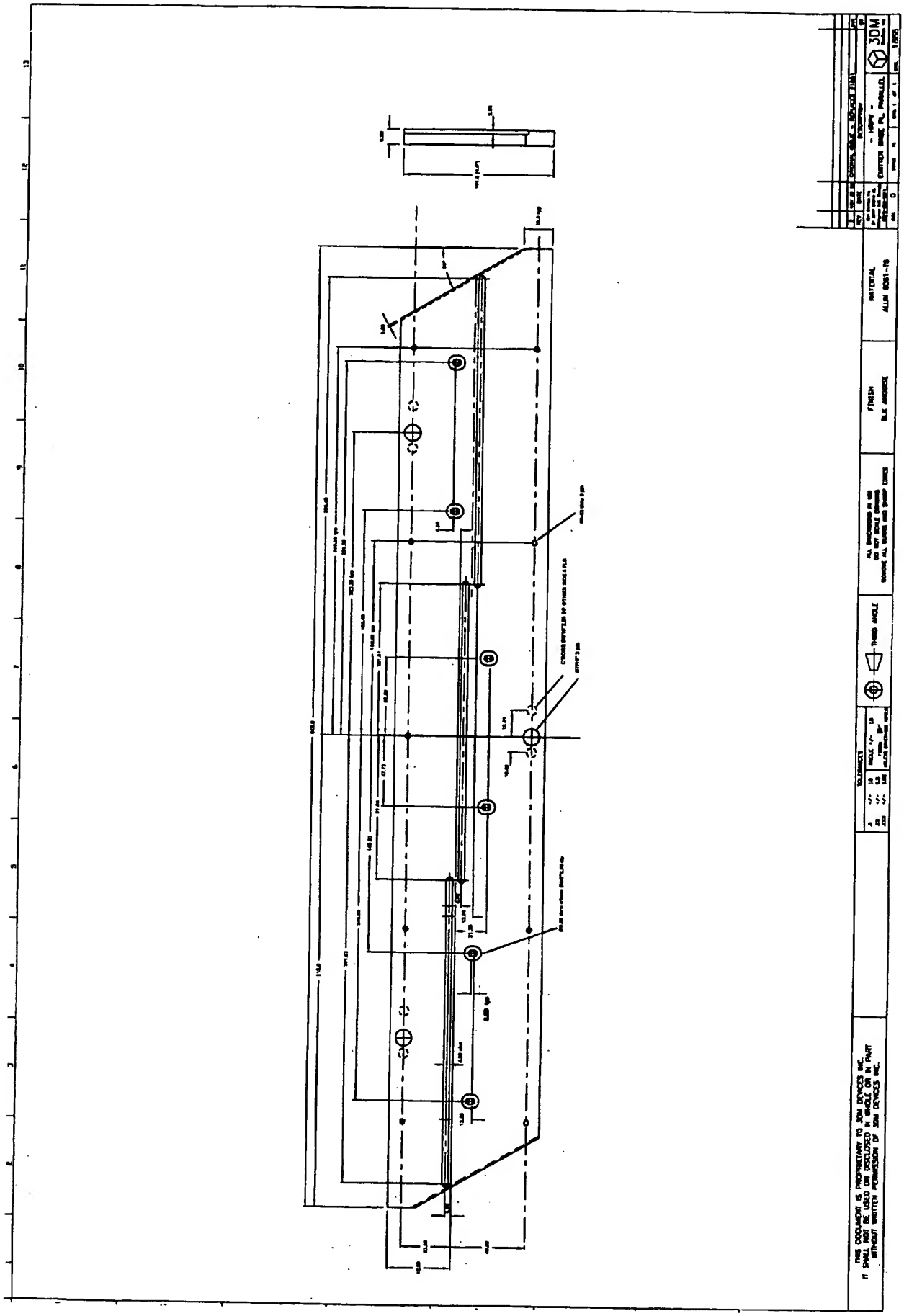
Goals	Hrpv gui gadgets function fully		
Testing	Play with gui gadgets		
	Click on controls very fast—the buffer usage should not keep rising		
Build 6			
Goals	Mask can be edited and uploaded		
Testing	Ensure mask uploads and downloads properly		
	Ensure mask works as expected on the incoming data		

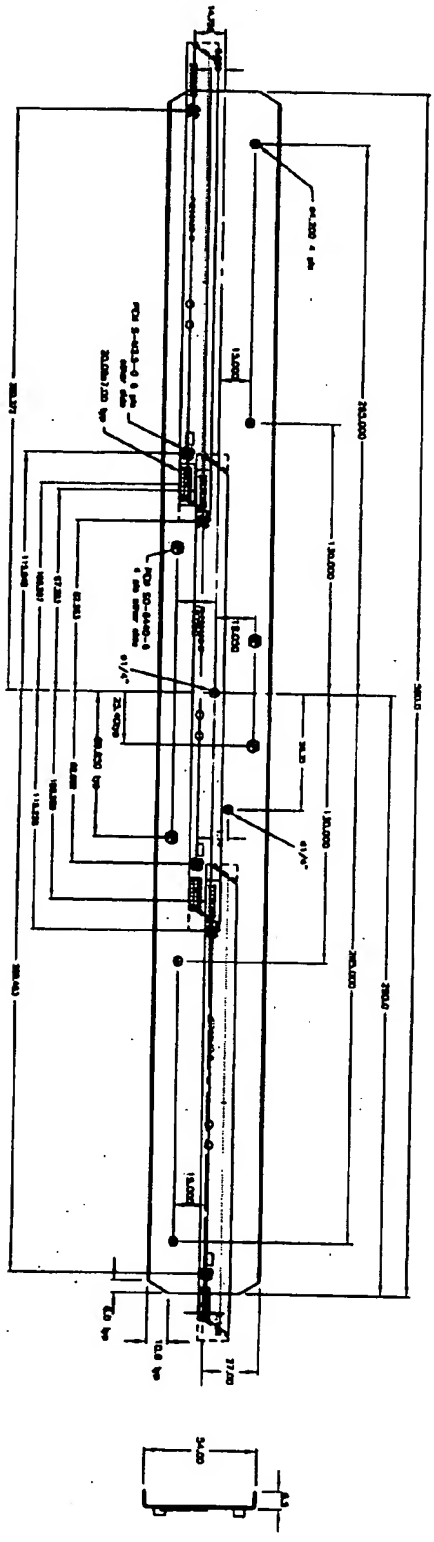
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

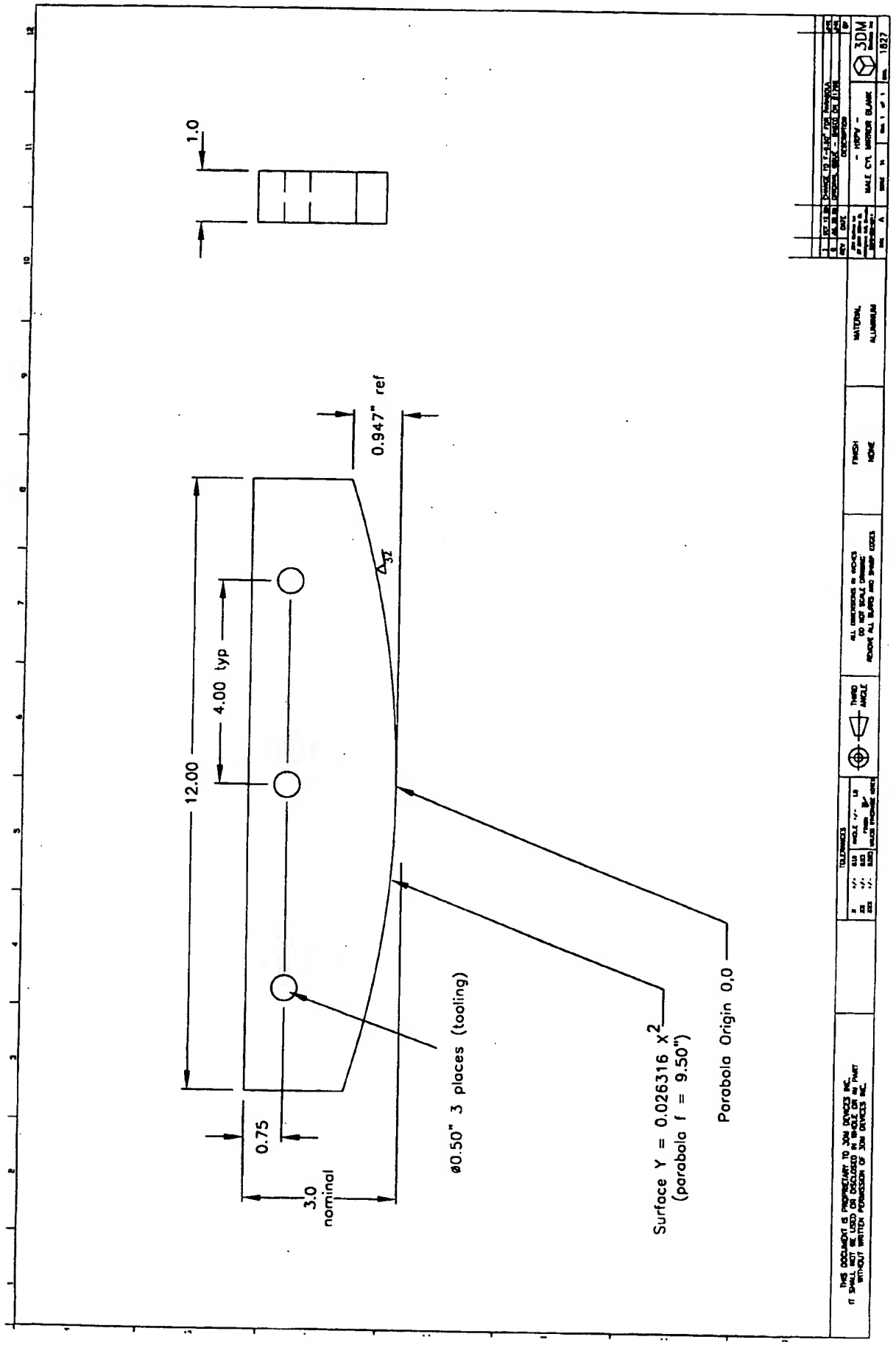
6.0 Pending Issues

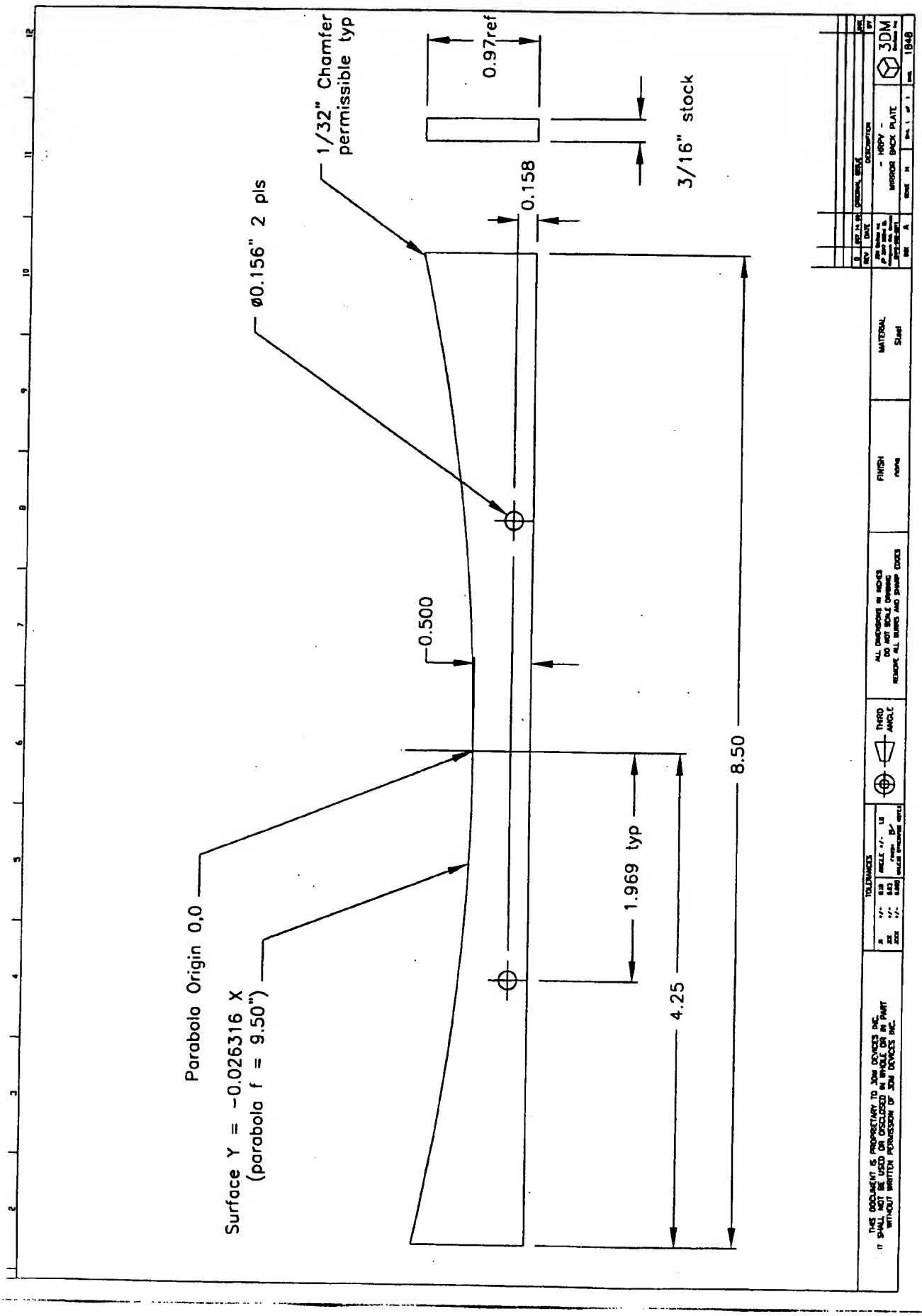
- When should the host ask for status?
- How will the host get information from the hub about errors?

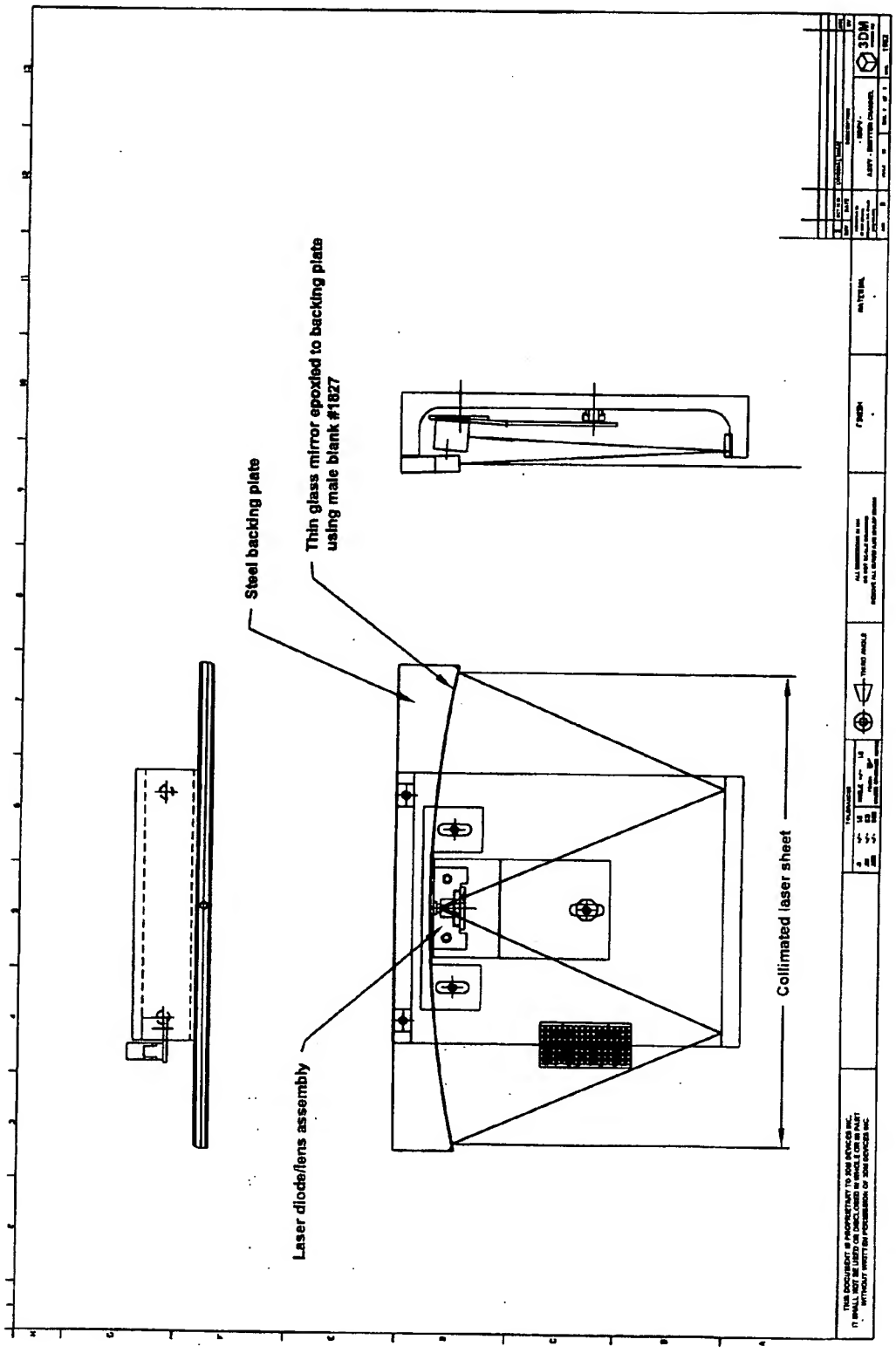
This document contains information proprietary to 3DM Devices Inc. Any disclosure, use or duplication of this document or of any of the information contained therein for other than the specific purpose for which it was disclosed is expressly prohibited except as 3DM Devices Inc. may otherwise agree to in writing.

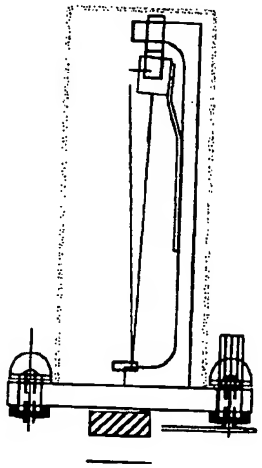


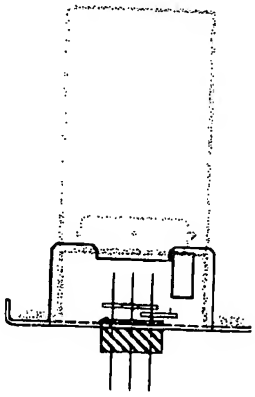



1.0
0.947" ref
12.00
4.00 typ
0.75
3.0
nominal
ø0.50" 3 places (tooling)
Surface Y = 0.026316 X2
(parabola f = 9.50")
Parabola Origin 0,0
- HSPV -
MALE CYL MIRROR BLANK
3DM
MATERIAL
ALUMINUM
FINISH
NONE
THIRD ANGLE
THIS DOCUMENT IS PROPRIETARY TO 3DM DEVICES INC.
IT SHALL NOT BE USED OR DISCLOSED IN WHOLE OR IN PART
WITHOUT WRITTEN PERMISSION OF 3DM DEVICES INC.

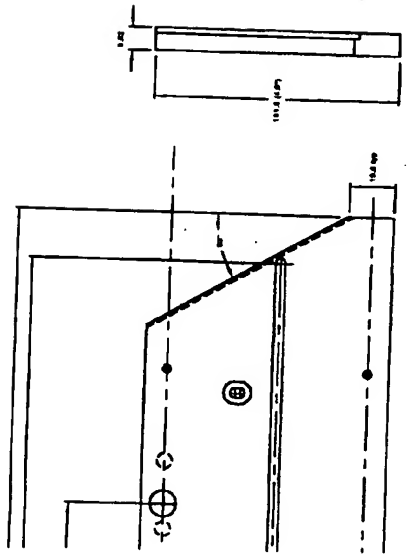

1/32" Chamfer
permissible typ
0.97ref
3/16" stock
0.158
Ø0.156" 2 pls
0.500
Parabolo Origin 0,0
Surface Y = -0.026316 X
(parabola f = 9.50")
1.969 typ
4.25
8.50
THIRD ANGLE
MATERIAL
Steel
FINISH
none
- HPPV -
MIRROR BACK PLATE
3DM
THIS DOCUMENT IS PROPRIETARY TO 3DM DEVICES INC.
IT SHALL NOT BE USED OR DISCLOSED IN WHOLE OR IN PART
WITHOUT WRITTEN PERMISSION OF 3DM DEVICES INC.


Steel backing plate
Thin glass mirror epoxied to backing plate using male blank #1827
Laser diode/lens assembly
Collimated laser sheet

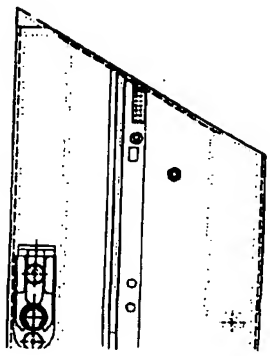




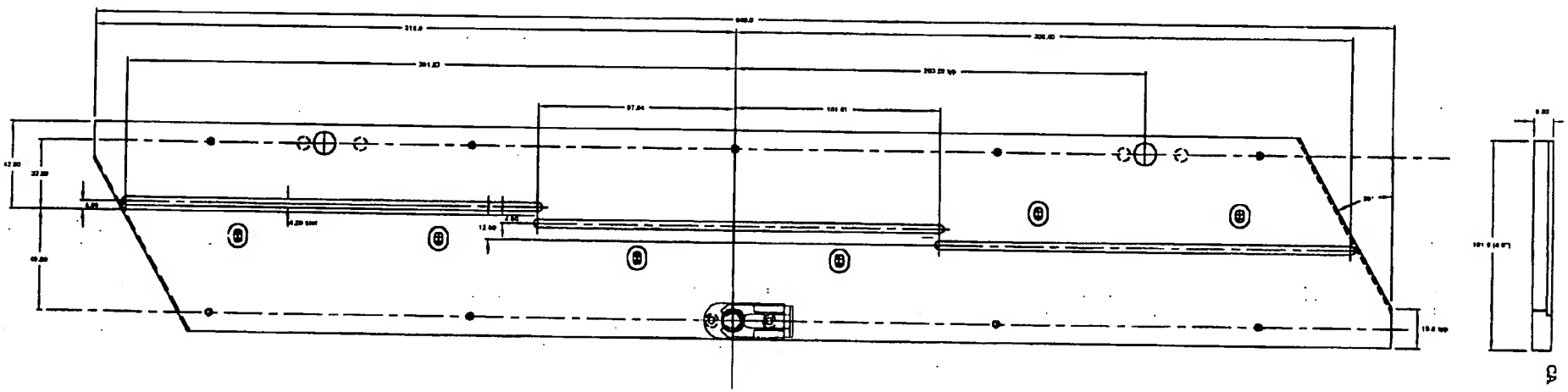
s Emitting out of paper



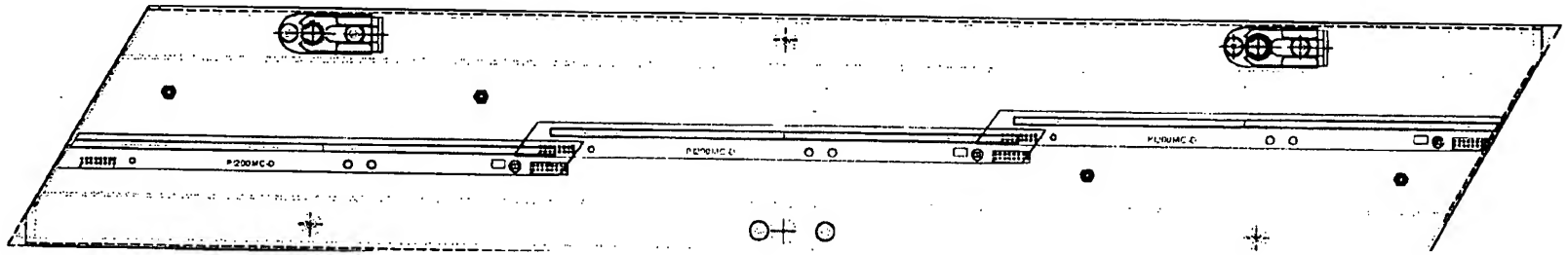
Ito detector element



Emitter - Lasers Emitting out of paper



Detector - Looking into detector element



CA 02334375 2001-02-02

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.